PAPER

Fault-Tolerant and Self-Stabilizing Protocols Using an Unreliable Failure Detector

Hiroyoshi MATSUI[†*], ***Nonmember***, **Michiko INOUE**[†], **Toshimitsu MASUZAWA**[†], ***and*** **Hideo FUJIWARA**[†], ***Regular Members***

SUMMARY We investigate possibility of fault-tolerant and self-stabilizing protocols (*ftss* protocols) using an unreliable failure detector. Our main contribution is (1) to newly introduce *k-accuracy* of an unreliable failure detector, (2) to show that *k-accuracy* of a failure detector is necessary for any *ftss* k-group consensus protocol, and (3) to present three *ftss* k-group consensus protocols using a k-accurate and weakly complete failure detector under the read/write daemon on complete networks and on $(n-k+1)$-connected networks, and under the central daemon on complete networks.

key words: *distributed algorithms, self-stabilization, fault-tolerance, failure detector, x-group consensus*

1. Introduction

Research on protocols that are both *fault-tolerant* and *self-stabilizing* is important to develop *truely* reliable distributed systems. A *self-stabilizing* protocol is a protocol that eventually achieves its intended behavior regardless of the initial network configuration. A self-stabilizing protocol tolerates any number of and any kind of *transient* faults in a sense that it can converge from any configuration resulted by transient faults if no further fault occurs for a sufficiently long period of time. On the other hand, a *t-fault-tolerant* protocol (for a *specific permanent* fault model) is a protocol that always achieves its intended behavior from a designated initial configuration regardless of at most t faults.

Gopal and Perry [1] first combined the concepts of fault-tolerance and self-stabilization. They consider the *general omission faults* (i.e., send and/or receive omission, and/or crashing), and presented a compiler that transforms a fault-tolerant protocol into a fault-tolerant and self-stabilizing protocol for a synchronous system. They also showed a fault-tolerant and self-stabilizing consensus protocol using unreliable failure detectors [2], [3] on asynchronous systems. Anagnostou and Hadzilacos [4] considered the crash faults. They defined a class of problems called *failure-sensitive* problems that includes the *counting* problem and the *leader election*, and showed that no 1-fault-tolerant and self-stabilizing protocol exists for the *failure-sensitive* problems. They also presented randomized 1-fault-tolerant and self-stabilizing protocol for the unique naming problem on ring networks. Masuzawa [5] defined the topology problem as a generalized problem of the counting problem. He considered the crash faults and presented a $(c-1)$-fault-tolerant and self-stabilizing protocol for the topology problem on c-connected networks under the assumption that each processor knows the neighbors' identifiers. He also showed that there exists no 1-fault-tolerant and self-stabilizing protocol using only either the neighbors' identifiers or the knowledge of connectivity. Beauquier and Kekkonen-Moneta [6] considered the crash fault and tried to clarify the problems for which there exist k-fault-tolerant and self-stabilizing protocols. They also presented 1-fault-tolerant and self-stabilizing protocols for some problems on ring networks.

In this paper, we consider the crash faults, and investigate possibility of fault-tolerant and self-stabilizing protocols using a failure detector. We extend an accuracy property of a failure detector and newly define a *k-accuracy* property, which guarantees that at least k correct processors are never suspected by any processors. We also define the *x-group consensus* problem, which requires correct processors to select common x correct processors. This problem is failure sensitive, and a generalized problem of the election problem. Our main results are (1) to show that *k-accuracy* of the failure detector is necessary for a fault-tolerant and self-stabilizing k-group consensus protocol, and (2) to present three $(n-k)$-fault-tolerant and self-stabilizing k-group consensus protocols which use a k-accurate and weakly complete failure detector; a space-unbounded protocol on complete networks under the read/write daemon, a space-unbounded protocol on $(n-k+1)$-connected networks under the read/write daemon, and a space-bounded protocol on complete networks under the central daemon, where n is the number of processors. Our protocols are based on the *checking and correction* technique, which is widely studied to transform protocols into self-stabilizing ones [7]–[9].

We treat two types of daemons, the read/write daemon and the central daemon. The two types of daemons are different in atomicity of an action of a processor: the read/write daemon assumes finer grain of atomicity. To classify influence of the difference on

Manuscript received July 14, 1999.
Manuscript revised March 6, 2000.

[†]The authors are with the Graduate School of Information Science, Nara Institute of Science and Technology (NAIST), Ikoma-shi, 630–0101 Japan.
[*]Presently, the author is with DDI Pocket, Japan.

the possibility of self-stabilization is interesting and has been investigated [10]. It is known that there exists a problem that is solvable under the central daemon but is unsolvable under the read/write daemon by self-stabilizing protocols [11], [12]. We have interest with relationship between such atomicity and *fault-tolerant and self-stabilizing* protocols. In this paper, we present only space-unbounded protocols under the read/write daemon, while we can present a space-bounded protocol under the central daemon.

Chandra et al. [2], [3] investigated what information about failures is necessary and sufficient for *fault tolerant* protocols to solve the consensus problem. They showed the weakest (i.e., necessary and sufficient) failure detector for *fault tolerant* consensus protocols. In this paper, we investigate what information about failures is necessary and sufficient for *fault-tolerant and self-stabilizing* protocols to solve the k-group consensus problem which is a generalized problem of the leader election. In short, our results show that k-accuracy is necessary and sufficient for *fault-tolerant and self-stabilizing* k-group consensus protocols.

We remark a "Γ-accurate" failure detector introduced by Guerraoui et al. [13] (independently of our work), where Γ is a subset of processors. The Γ-accuracy is motivated by the observation that processors suspected to be crashed should be restricted when the system is partitioned. Therefore, it specifies a set Γ of processors that are not mistakenly suspected as crashed processors. Our k-accuracy has a quite different motivation. The k-accuracy specifies the number of the processors mistakenly suspected to be crashed. Network partitioning is avoided by requiring $(n-k+1)$-connectivity in this paper.

The rest of this paper is organized as follows. Section 2 and Sect. 3 present the computation model and several definitions. Section 4 shows the necessity of the k-accuracy of the failure detector for fault-tolerant and self-stabilizing k-group consensus protocols. Three $(n-k)$-fault-tolerant and self-stabilizing k-group consensus protocols are presented in Sect. 5.

2. Preliminaries

2.1 Model

A network $N = (P, L)$ consists of a set $P = \{p_1, p_2, \cdots, p_n\}$ of processors and a set L of communication links (simply called links), where each link is a pair of distinct two processors. If $(p_i, p_j) \in L$ holds, then p_i and p_j are called *neighbors*. A processor is a state machine. Each processor p_i has a unique identifier id_i, drawn from some totally ordered set. We adopt the *link-register* model introduced in [14]. Two neighbors p_i and p_j communicate using two *shared communication registers* (simply called registers) $R_{i,j}$ and $R_{j,i}$. The register $R_{i,j}$ can be written only by p_i and read only by p_j. The register $R_{i,j}$ is called an *output* register of p_i and an *input* register of p_j.

A *configuration* of a network is a vector of processor states and resister contents. Let m be the number of the registers, and let S_i be the set of states of processor p_i and Σ_j be the set of symbols that can be stored in the j^{th} register[†]. The set C of all possible configurations is

$$C = S_1 \times S_2 \times \cdots \times S_n \times \Sigma_1 \times \Sigma_2 \times \cdots \times \Sigma_m.$$

A protocol is a collection of algorithms, one for each processor. Activity of processors is managed by a *daemon*. Whenever the daemon activates a processor, the processor executes an atomic step of its algorithm. In this paper, we use two types of daemons. The *central* daemon (C daemon, in short) activates one processor at a time, and the atomic step of a processor consists of (1) reading all its input registers, (2) changing its state, and (3) writing all its output registers. The *read/write* daemon (R/W daemon, in short) activates one processor at a time, and the atomic step of a processor consists of (1) either reading one of its input registers or writing one of its output registers (but not both), and (2) changing its state.

An execution $E = c_0, c_1, c_2 \cdots$ of a protocol A is an infinite sequence of configurations, where each c_{h+1} $(h \geq 0)$ is reachable from c_h by a single atomic step of some processor according to A. Configuration c_0 is called an *initial configuration* of E. We assume, for each $h \geq 0$, an atomic step a which changes the configuration from c_h to c_{h+1} is uniquely determined. That is, the execution implicitly defines a sequence of atomic steps. Note that any non-empty suffix of any execution is also an execution.

A processor is *faulty* if it does not follow the protocol. We consider only *crash faults* of processors: a faulty processor stops prematurely and does nothing from that point on, however, it behaves correctly before stopping. In the model of the state machine, occurrence of the crash fault is modeled as execution of a special step called a *crash step*. The crash step changes the processor state into a special state, *crash state*, and has no effect on registers. In the crash state, no further step can be executed. The crash step can be executed at any state except for the crash state.

Given an execution E of a protocol A, let $\mathcal{F}(E)$ denote a set of faulty processors (i.e. those in the crash state after some point) and $\mathcal{C}(E)$ $(= P - \mathcal{F}(E))$ denote a set of correct processors. If every processor in $\mathcal{C}(E)$ makes infinitely many steps in E, then E is called a *fair* execution. We consider only a fair execution in this paper, and simply use the term an *execution* for a fair execution. We assume that a network is asynchronous: there is no assumption on the number of

[†]For convenience, we assume a total order on the registers. This order is used only to describe the configuration, and cannot be used in designing protocols.

steps each processor executes in any prefix of an execution. Note that processor faults cannot be detected in such an asynchronous network since it is impossible to determine whether a processor has actually crashed or is only "very slow".

A *problem* specifies the required behavior of processors. Formally we define a problem to be a set of *legal executions*, which are executions satisfying the problem requirement. A problem Π on a network $N = (P, L)$ with a set $F \subseteq P$ of faulty processors is defined by a set of legal executions denoted by $L_\Pi(N, F)$. Let ${}_S L_\Pi(N, F)$ denote a set of all non-empty suffixes of legal executions in $L_\Pi(N, F)$.

A t-fault-tolerant (*t-ft*) protocol for a problem Π is a protocol whose any execution starting from a designated initial configuration is legal for Π regardless of at most t faulty processors. The designated initial configuration is a configuration in which each processor is in a prescribed initial state and each register contains a prescribed symbol as its initial value.

Definition 1: Let $\mathcal{N}$ be a family of networks, t be a non-negative integer, and Π be a problem. A protocol A is a *t-fault-tolerant* (*t-ft*) protocol for Π in $\mathcal{N}$, if any execution E of A such that E starts from the designated initial configuration in any network N ($\in \mathcal{N}$) and satisfies $|\mathcal{F}(E)| \le t$ is in $L_\Pi(N, \mathcal{F}(E))$.

A *t-fault-tolerant* and *self-stabilizing* (*t-ftss*) protocol for a problem Π is a protocol such that its any execution E converges to some legal execution L of Π (i.e., E and L have a common suffix) regardless of its initial configuration and at most t faulty processors.

Definition 2: Let $\mathcal{N}$ be a family of networks, t be a non-negative integer, and Π be a problem. A protocol A is a *t-fault-tolerant* and *self-stabilizing* (*t-ftss*) protocol for Π in $\mathcal{N}$, if any execution E of A such that E starts from any configuration in any network N ($\in \mathcal{N}$) and satisfies $|\mathcal{F}(E)| \le t$ has a suffix E' in ${}_S L_\Pi(N, \mathcal{F}(E))$.

Usually, the above definition of stabilization is called *pseudo-stabilization* [15], and stabilization is defined by reachability to some legitimate configuration and closure of a set of the legitimate configurations. However, we consider the crash faults that can occur out of control of the protocol, and deal with a *failure-sensitive* problem that changes the legal executions by occurrence of faults. Therefore, it is impossible to design a protocol which guarantees the closure of a set of the legitimate configurations, and we adopt the definition of the pseudo-stabilization.

In this paper, we make some additional assumption on a network. First, we assume that each processor initially knows the identifiers of its neighbors as well as its own identifier and this knowledge cannot be corrupted by transient faults. That is, every processor knows accurate identifiers of itself and its neighbors at any configuration. In the case of complete networks, this means that every processor initially knows accurate identifiers of all processors.

2.2 x-Group Consensus

Anagnostou and Hadzilacos [4] showed that *failure-sensitive* problems, including the leader election problem, has no 1-*ftss* protocol. In this paper, we define and consider the *x-group consensus* problem as a generalized problem of the leader election problem, where x is a positive integer. The x-group consensus problem requires that correct processors select common x correct processors. This problem is very attractive since it can be available such an universal solution that first we select x correct processors and then the selected x processors cooperatively solve a given problem. The x-group consensus problem is failure-sensitive, and there exists no 1-*ftss* x-group consensus protocol.

Definition 3 (x-group consensus problem): Let $N = (P, L)$ be a network. Assume that each processor p_i has a variable $Active_i$ representing a set of processor identifiers[†]. An execution $E = c_0, c_1, \cdots$ is legal for x-group consensus problem Π (i.e., $E \in L_\Pi(N, \mathcal{F}(E))$), iff there exist a set P' ($\subseteq \mathcal{C}(E)$) of x correct processors (i.e., $|P'| = x$) and an integer h_0 ($h_0 \ge 0$) such that, in any configuration c_h ($h \ge h_0$), $Active_i = \{id_j | p_j \in P'\}$ holds for any correct processors p_i ($\in \mathcal{C}(E)$).

2.3 Failure Detector

We use an *unreliable failure detector* introduced by Chandra and Toueg [2]. The failure detector consists of a collection of failure detecting processes, one for each processor. The failure detecting process for a processor p_i repeatedly suspects faulty processors except for p_i and manages p_i's local variable FP_i representing an identifier set of suspected processors. The change of the value of FP_i can be modeled by a change of the state of p_i. For an execution $E = c_0, c_1, \cdots$, let $FP_i^{E,h}$ denote a value of FP_i in configuration c_h. If $id_j \in FP_i^{E,h}$ holds, we say that p_i suspects p_j in c_h.

A failure detector is specified by two properties, *completeness* and *accuracy*. Chandra and Toueg [2] considered two completeness properties and four accuracy properties. Strong (resp. weak) completeness guarantees that every faulty processor is eventually suspected by *all* (resp. *some*) correct processors.

Definition 4 (strong completeness): A failure detector is *strongly complete* if, for any execution E, there exists some h_0 such that, for any correct processor p_i ($\in \mathcal{C}(E)$) and any h ($\ge h_0$), $\mathcal{F}(E) \subseteq FP_i^{E,h}$ holds.

Definition 5 (weak completeness): A failure detector is *weakly complete* if, for any execution E and any faulty

[†]For convenience, we use variables and a program to represent a processor state and a state transition function.

processor p_i ($\in \mathcal{F}(E)$), there exist some correct processor p_j ($\in \mathcal{C}(E)$) and some h_0 such that, for any h ($\geq h_0$), $p_i \in FP_j^{E,h}$ holds.

Accuracy restricts the mistakes of a failure detector. In [3], four accuracy properties, *strong accuracy*, *weak accuracy*, *eventually strong accuracy* and *eventually weak accuracy* are defined. Intuitively, strong accuracy guarantees that no processor is suspected before it crashes, and weak accuracy guarantees that some correct processes is never suspected. Eventually strong (resp. weak) accuracy means that strong (resp. weak) accuracy holds *eventually.* In this paper, we consider some hierarchy between strong and weak accuracy. We newly define *k-accuracy*, which guarantees that at least k correct processors are not suspected by any processors. Clearly, 1-accuracy is equivalent to the weak accuracy, and any k-accuracy is weaker than the strong accuracy since it guarantees that any correct processors is never suspected.

Definition 6 (k-accuracy): A failure detector is *k-accurate* if the following holds: for any execution E satisfying $|\mathcal{C}(E)| \geq k$, there exists a set P' ($\subseteq \mathcal{C}(E)$) of k correct processors such that, for any processor p_i and any integer h (≥ 0), $P' \cap FP_i^{E,h} = \emptyset$ holds.

We can also consider *eventually k-accuracy* property, which means that the k-accuracy holds *eventually.* However, we does not consider such a property, since self-stabilizing protocols are required to *eventually* achieve their intended behavior, therefore, if we consider an execution only after the k-accuracy holds, it can be considered that the eventually k-accuracy is equivalent with the k-accuracy.

3. Necessity of k-Accuracy for k-Group Consensus

In this section, we show that the k-accuracy of the failure detector is necessary for 1-*ftss* k-group consensus protocols. We show that there is no 1-*ftss* k-group consensus protocol for the k-group consensus problem which uses a $(k-1)$-accurate and strongly complete failure detector. Since a failure detector is specified by completeness and accuracy, and strong completeness is the strongest with respect to completeness, this result implies that k-accuracy of the failure detector is necessary for 1-*ftss* k-group consensus protocols, and hence, for any *ftss* k-group consensus protocols.

First, we define some notations. Let c and c' be configurations. Let $c \overset{i}{\bowtie} c'$ denote a configuration that is identical to c except that p_i's state is the same as in c'. Let ID be a set of identifiers. A configuration c is *ID-consensus* if $Active_i = ID$ for every processor p_i which is not in the crash state. Let C_k denote a set of all ID-consensus configurations such that the size of ID is k.

Theorem 1: There exists no 1-*ftss* protocol for the k-group consensus problem under the C daemon, even if it can use a $(k-1)$-accurate and strongly complete failure detector.

(Proof) Assume that a protocol A is a 1-*ftss* protocol for the k-group consensus problem using a $(k-1)$-accurate and strongly complete failure detector in some network family. We consider the following execution $E = c_0, c_1, \cdots$ where $\mathcal{F}(E) = \emptyset$ and every processor suspects all the processors except for some $k-1$ processors and itself. Let FP be a set of $n-k+1$ identifiers such that $FP_i^{E,h} = FP - \{id_i\}$ for any $p_i \in P$ and any h (≥ 0).

First, the daemon activates all processors until some ID-consensus configuration c_{h_1} in C_k is reached. Since $|ID| = k$ and $|FP| = n-k+1$, there exists some id_i such that $id_i \in ID \cap FP$. We temporarily assume that p_i in the crash state c_{h_1}. Since A is a 1-*ftss* protocol, the daemon can lead the network to some ID'-consensus configuration c'_{h_2} in C_k where $id_i \notin ID'$. The processor p_i does nothing from c_{h_1} to c'_{h_2}, and the other processors cannot distinguish whether p_i has crashed or is just slow. Therefore, steps from c_{h_1} to c'_{h_2} are possible to occur if p_i is actually correct.

Now consider the case where p_i is a correct processor. In this case the daemon can leads the network to the configuration $c_{h_2} = c'_{h_2} \overset{i}{\bowtie} c_{h_1}$ by activating the processor except for p_i. In c_{h_2}, $Active_i = ID$ and $Active_j = ID' \neq ID$ for any j ($j \neq i$), therefore, $c_{h_2} \notin C_k$. Since A is 1-*ftss* protocol, some configuration c_{h_3} in C_k is reached again.

By repeating the above strategy, the daemon can schedule processor steps so that configurations not in C_k appear infinitely often in E. However, A is 1-*ftss* protocol, and therefore, E has a suffix consisting of only configurations in C_k. A contradiction occurs. $\square$

Since the R/W daemon has smaller atomicity than C daemon, the R/W daemon can activate processros in the same way as the C daemon. Thus, impossibility results for the C daemon holds for the R/W daemon, and the following corollary holds.

Corollary 1: There exists no 1-*ftss* protocol for the k-group consensus problem under the R/W daemon, even if it can use a $(k-1)$-accurate and strongly complete failure detector.

4. $(n-k)$-*ftss* k-Group Consensus Protocols

4.1 Overview

We present the following three $(n-k)$-*ftss* k-group consensus protocols.

- *RWKP* : a space-unbounded protocol under the R/W daemon in complete networks.

- $RWKP'$: a space-unbounded protocol under the R/W daemon in $(n-k+1)$-connected networks.
- CKP : a space-bounded protocol under the C daemon in complete networks.

First, we describe the common overview to all protocols. In the description of the protocols in Fig. 1, Fig. 2 and Fig. 3, $read_{i,j}(x_i)$ denotes that p_i reads its input register $R_{j,i}$ and stores the value to its local variable x_i, and $write_{i,j}(x_i)$ denotes that p_i writes the value of its local variable x_i to its output register $R_{i,j}$. If the variable x_i is partitioned into some fields $x_i.a$, $x_i.b, \cdots$, we refer the corresponding fields of $R_{i,j}$ as $R_i.a$, $R_i.b, \cdots$. Let S be an identifier set. Let $pick_k(S)$ denote a function returning the smallest k identifiers in S. Note that a variable FP_i denote an identifier set of suspected processors and it is under the control of the failure detecting process for p_i. In this subsection, we present $(n-k)$-*ftss* protocols, and therefore, we consider only such an execution E that $\mathcal{F}(e) \le n-k$ holds.

Our three protocols are based on a $(n-k)$-*ft* protocol KP in complete networks under the R/W daemon (Fig. 1). The protocol KP uses a k-accurate and weakly complete failure detector. The protocol is correct if all $Active_i$ $(= pick_k(\{id_1, \cdots, id_n\} - sus_i))$ of

```
var
  $sus_i, Active_i$: set of processor ids;
  $rsus (1 \le j \le n)$: set of processor ids;
begin
  $sus_i := \emptyset$; /* initialization */
  repeat forever do
    $sus_i := sus_i \cup FP_i$;
    for each $j$ $(1 \le j \le n, j \ne i)$ do
      $read_{i,j}(rsus)$;
      $sus_i := sus_i \cup rsus$;
    $Active_i := pick_k(\{id_1, \cdots, id_n\} - sus_i)$;
    for each $j$ $(1 \le j \le n, j \ne i)$ do
      $write_{i,j}(sus_i)$;
end
```

Fig. 1 $(n-k)$-*ft* protocol KP: code for p_i.

```
var
  $sus_i$, $Active_i$, $sdata.sus$, $rdata.sus$
    : set of processor ids;
  $sdata.vn$, $rdata.vn$ : integer;
begin
  repeat forever do
    $sus_i := sus_i \cup FP_i$;
    for each $j$ $(1 \le j \le n, j \ne i)$ do
      $read_{i,j}(rdata)$;
      if $rdata.vn = vn_i$ then
        $sus_i := sus_i \cup rdata.sus$;
        if $|sus_i| > n-k$ then
          $sus_i := \emptyset$; $vn_i := vn_i + 1$;
        else
          $Active_i := pick_k(\{id_1, \cdots, id_n\} - sus_i)$;
      else if $rdata.vn > vn_i$ then
        $vn_i := rdata.vn$; $sus_i := rdata.sus$;
    $sdata.sus := sus_i$; $sdata.vn := vn_i$;
    for each $j$ $(1 \le j \le n, j \ne i)$ do
      $write_{i,j}(sdata)$
end
```

Fig. 2 $(n-k)$-*ftss* protocol $RWKP$:code for p_i.

```
var
  $sus_i$, $sdata_{i,j}.sus$, $rdata_{j,i}.sus$, $Active_i$, $Rcv_i$
    : set of processor ids;
  $sdata_{i,j}.F$, $rdata_{j,i}.F$: boolean;
      /* flag for communication mechnism */
  $sdata_{i,j}.R$, $rdata_{j,i}.R$: boolean; /* reset request */
  $mode_i$: NORMAL or RESET;
begin
  repeat forever do
    /* receive messages */
    $Rcv_i := \emptyset$;
    for each $j$ $(j \ne i)$ do
      $read_{i,j}(rdata_{j,i})$;
      /* select newly received messages */
      if $first\_read(R_{i,j}) = true$
        then $Rcv_i := Rcv_i \cup \{j\}$;

    /* update $sus_i$ */
    if there exists $j \in Rcv_i$ s.t. $rdata_{j,i}.R = true$
      then /* case: reset-request */
        $sus_i := \emptyset$; $mode_i := NORMAL$;
      else /* case: normal message */
        /* update a total suspicion */
        $sus_i := sus_i \cup \bigcup_{j \in Rcv_i} rdata_{j,i}.sus \cup FP_i$;
        if $|sus_i| > n-k$
          then /* inconsistency is detected */
            /* reset itself */
            $sus_i := \emptyset$; $mode_i := RESET$;
          else
            /* select $k$ correct processros */
            $Active_i := pick_k(\{id_1, \cdots, id_n\} - sus_i)$;
            $mode_i := NORMAL$;

    /* set messages for processors read privious messages. */
    for each $j \in Rcv_i$ do
      $sdata_{i,j}.sus := sus_i$; $sdata_{i,j}.R := false$;
      /* for communication mechanism */
      $sdata_{i,j}.F := (sdata_{i,j}.F + 1)$ mod 2;

    /* update messages if reset mode */
    if $mode_i = RESET$ /* reset mode */
      then
        /* set reset-requests for all processors */
        for each $j$ $(j \ne i)$ do
          $sdata_{i,j}.R := true$;  /* reset-request */

    /* send messages */
    for each $j$ $(j \ne i)$ do
      $write_{i,j}(sdata_{i,j})$;
end.
```

Fig. 3 $(n-k)$-*ftss* protocol CKP: code for p_i.

correct processors converge to the same set of k correct processors. To show this convergence, we prove the convergence of a variable sus_i. A variable sus_i represents a set of processor identifiers which p_i itself suspects or p_i knows some processor suspects. We call sus_i a *total suspicion* of p_i. We can observe that every total suspicion monotonically increases with respect to the inclusion relation '$\subseteq$', any correct processor p_i's total suspicion will be included by any other correct processor p_j's total suspicion after sufficient number of steps, and a total suspicion of every correct processor is bounded from above by a set of all identifiers. Therefore, all total suspicions of correct processors converge to the same set sus of identifiers. From the k-accuracy and the weak completeness of the failure detector, this sus includes all faulty processors and never includes at least k correct processors if there exist at least k correct processors (i.e., at most $n-k$ faulty processors). Therefore, all $Active_i$ of correct processors converge to the same set of k correct processors. This means that KP is an $(n-k)$-*ft* k-group consensus protocol.

Now, we extend $(n-k)$-*ft* protocol KP to $(n-k)$-*ftss* protocols. For *ftss* protocols, we can assume nothing on the initial total suspicion. If some sus_i initially includes many correct processor identifiers, the size of sus_i may exceed $n-k$. This is inconsistent with the k-accuracy of the failure detector. In our *ftss* protocols, each p_i checks such *inconsistency* (i.e., $|sus_i| > n-k$) whenever it updates the value of sus_i. If p_i detects the inconsistency, p_i resets its state (sets its total suspicion empty) and attempts to reset a network configuration (set the network to some configuration reachable from the designated initial configuration of KP). If the network configuration can be reset, the k-group consensus problem can be solved.

Note that the protocol KP has infinite iterations and any processors does not know when the variable $Active_i$ converges. This is natural because the convergence period of $Active_i$ depends on both time when processors crash and suspicions of failure detectors.

4.2 Protocols under the R/W Daemon

We present an $(n-k)$-*ftss* protocols for the k-group consensus problem using a k-accurate and weakly complete failure detector under the R/W daemon. First, we present a protocol $RWKP$ (Fig. 2) in complete networks, and then extend it to be applicable to $(n-k+1)$-connected networks.

In $RWKP$, each processor p_i uses a variable vn_i representing the version number of its local suspicion sus_i. Each processor exchanges the total suspicion with other processors. When p_i reads the total suspicion $rdata.sus$ with version number $rdata.vn$ from its input register, p_i updates its total suspicion as follows: (1) if $rdata.vn > vn_i$, p_i resets its total suspicion and sets sus_i to $rdata.sus$, (2) if $rdata.vn = vn_i$, p_i adds $rdata.sus$ to sus_i, and if it becomes inconsistent, i.e., $|sus_i| > n-k$, p_i resets its total suspicion and increments its version number by one, and (3) if $rdata.vn < vn_i$, p_i ignores $rdata.sus$ and does nothing.

To prove the correctness of $RWKP$, we must show the convergence of the variables $Active_i$ of all correct processors. This convergence is derived directly from the convergence of the total suspicions of all correct processors. To show this, we use the following lemma. It shows conditions for that all total suspicions converge to the same set which includes all faulty processors. For an execution $E = c_0, c_1, \cdots$, let $v^{E,h}$ denote a value of a variable v in a configuration c_h.

Lemma 1: Consider any protocol in which each processor p_i has a variable sus_i of an identifier set and uses a weakly complete failure detector. Let $E = c_0, c_1, \cdots$ be an execution of the protocol. If the following four conditions hold, there exist some set sus and some g_0 such that $\mathcal{F}(E) \subseteq sus$ and, for any p_i ($\in \mathcal{C}(E)$) and any g ($\geq g_0$), $sus_i^{E,g} = sus$ holds.

(1) There exists a set S of identifiers such that $sus_i^{E,h} \subseteq S$ holds for any p_i ($\in \mathcal{C}(E)$) and any h.
(2) For any p_i ($\in \mathcal{C}(E)$) and any h and h' ($h \leq h'$), $sus_i^{E,h} \subseteq sus_i^{E,h'}$ holds.
(3) For any p_i and p_j ($p_i, p_j \in \mathcal{C}(E)$) and any h, there exists some h' such that $sus_i^{E,h} \subseteq sus_j^{E,h'}$.
(4) For any sus_i ($p_i \in \mathcal{C}(E)$) and any h ($h > 0$), $FP_i^{E,h-1} \subseteq sus_i^{E,h}$ holds.

(Proof) For every correct processor, condition (1) means sus_i has an upper bound (w.r.t. '$\subseteq$'), and condition (2) means sus_i monotonically increases. Therefore, every sus_i ($p_i \in \mathcal{C}(E)$) converges to some set $final\_sus_i$. The condition (3) implies, for any correct processors p_i and p_j, $final\_sus_i \subseteq final\_sus_j$ and $final\_sus_j \subseteq final\_sus_i$, therefore, $final\_sus_i = final\_sus_j$ holds. Therefore, there exist some set sus and some g_0 such that, for any p_i ($\in \mathcal{C}(E)$) and any g ($\geq g_0$), $sus_i^{E,g} = sus$ holds. Moreover, by (2) and (4), $FP_i^{E,h-1} \subseteq final\_sus_i$, and $\bigcup_{p_i \in \mathcal{C}(E)} FP_i^{E,h-1} \subseteq sus$ hold for any h ($h > 0$). By the weak completeness, $\mathcal{F}(E) \subseteq sus$ holds. □

Now we show the correctness of $RWKP$.

Lemma 2: For any execution $E = c_0, c_1, \cdots$ of $RWKP$ under the R/W daemon, there exists an integer vn such that $vn_i^{E,h} \leq vn$ holds for any i and h.

(Proof) Let max be the maximum version number appears in c_0. Assume that the lemma does not hold, i.e., some version numbers have no upper bound. Let c_g the first configuration in which some version number becomes no smaller than $max+2$. Let p_i be a processor such that $vn_i^{E,g} = max'$ ($\geq max+2$). In c_{g-1}, no version number is more than $max+1$, therefore, in a step from c_{g-1} to c_g, p_i ought to detect the inconsistency $|sus_i| > n-k$ and increment vn_i from $max'-1$ to max'. Let sus' denote the value of this inconsistent sus_i. In $RWKP$, every total suspicion is computed from the total suspicions with the same version number, therefore, p_i computes sus' from the total suspicions with version number $max'-1$. Since $max'-1 > max$, every processor with version number $max'-1$ has reset its total suspicion at least once. This implies that every total suspicion with version number $max'-1$ includes only the identifiers suspected by the failure detector. That is, $sus' \subseteq \bigcup_{p_i\in P, 0\le h<g} FP_i^{E,h}$ holds. However, $|\bigcup_{p_i\in P, 0\le h<g} FP_i^{E,h}| \le n-k$ holds by the $k-accuracy$, a contradiction occurs. □

Theorem 2: If the failure detector is k-accurate and weakly complete, the protocol $RWKP$ is an $(n-k)$-*ftss* k-group consensus protocol in complete networks under the R/W daemon.

(Proof) Consider any execution $E = c_0, c_1, \cdots$ of $RWKP$. By Lemma 2, there exists the maximum value max_vn of version numbers of correct processors appear in E. Let c_h be a configuration in which $vn_i^{E,h} = max\_vn$ for some correct processor p_i. Since each version number never decreases, $vn_i^{E,h'} = max\_vn$ holds for any $h' \ge h$. Fairness of executions guarantees that, for any correct processor p_j, p_i writes max_vn to the register $R_{i,j}$ as a value of vn_i, and after then p_j reads $R_{i,j}$. After p_j reads the value max_vn, vn_j becomes at least max_vn. Since max_vn is the maximum value of version numbers of correct processors, vn_j becomes max_vn and remains max_vn after that. That is, E has some suffix $E' = c'_0, c'_1, \cdots$ such that $vn_j^{E',h} = max\_vn$ for any correct processor p_j and any $h \ge 0$.

In E', any correct processor never increments its version number, therefore, it never resets its total suspicion. Now we show that Lemma 1 can be applied for E'. (1) Clearly, every $sus_i^{E',h} \subseteq \{id_i | p_i \in P\}$ holds for any $p_i \in \mathcal{C}(E')$ and h. (2) $sus_i^{E',h} \subseteq sus_i^{E,h'}$ holds for any $p_i \in \mathcal{C}(E')$ and any h and h' $(h \le h')$. (3) For any p_i and p_j $(p_i.p_j \in \mathcal{C}(E'))$ and any h, there exists some $\bar{h}$ and h' $(h \le \bar{h} \le h')$ such that p_i writes $sus_i^{E',\bar{h}}$ $(\supseteq sus_i^{E',h})$ to $R_{i,j}$ and then p_j reads $sus_i^{E',\bar{h}}$ from $R_{i,j}$ and sets $sus_j^{E',h'}$ $(\supseteq sus_i^{E',h})$. Finally, (4) for any sus_i $(p_i \in \mathcal{C}(E'))$ and any h $(h>0)$ $FP_i^{E',h-1} \subseteq sus_i^{E',h}$ holds. By the above (1), (2), (3) and (4), and the facts of $\mathcal{C}(E') = \mathcal{C}(E)$ and $\mathcal{F}(E') = \mathcal{F}(E)$, there exist some set sus and some g_0 such that $\mathcal{F}(E) \subseteq sus$ and, for any $p_i \in \mathcal{C}(E)$ and g $(\ge g_0)$, $sus_i^{E',g} = sus$ hold. Since any correct processor $(\in \mathcal{C}(E))$ never resets its total suspicion in E', $|sus| \le n-k$ holds. Let $Active$ be the value of $pick_k(\{id_1, id_2, \cdots, id_n\} - sus)$. Then, $|Active| = k$ and $Active \subseteq \{id_j | p_j \in \mathcal{C}(E)\}$ hold. For any $p_i \in \mathcal{C}(E)$ and any $g \ge g_0$, $sus_i^{E',g} = sus$ holds, and therefore, and $Active_i^{E',g} = Active$ holds. Since E' is a suffix of E, this implies that E is a legal execution for the k-group consensus problem. □

The protocol $RWKP$ in complete networks can be extended to an $(n-k)$-*ftss* protocol in any $(n-k+1)$-connected networks. Masuzawa [5] proposed an $(n-k)$-*ftss* topology protocol in $(n-k+1)$-connected networks under the assumption that each processor initially knows the identifiers of its neighbors. In an execution of the topology protocol, each processor eventually obtains the network topology including an accurate set of identifiers of all processors. Now consider the composite protocol $RWKP'$ of $RWKP$ and the topology protocol. In $RWKP'$, each processor alternatively executes a step of $RWKP'$ and a step of the topology protocol. The differences between $RWKP$ and $RWKP'$ are (1) each processor initially knows an accurate set of identifiers of all processors in $RWKP$, and (2) any two processors can directly communicate via the registers between them in $RWKP$. These are resolved as follows. (1) In any execution of $RWKP'$, each processor eventually obtains an accurate set of identifiers of all processors, and (2) any two correct processors have a path between them consisting of only correct processors, and fairness of executions guarantees that every total suspicion is propagated through the path unless it meets with a larger version number. Therefore, $RWKP'$ is an $(n-k)$-*ftss* k-group consensus protocol in $(n-k+1)$-connected networks.

Corollary 2: If the failure detector is k-accurate and weakly complete, the protocol $RWKP'$ is an $(n-k)$-*ftss* k-group consensus protocol in $(n-k+1)$-connected networks under the R/W daemon.

4.3 Protocol under the C Daemon

The protocol $RWKP$ and $RWKP'$ are space-unbounded, since they use an unbounded variable vn_i. In this subsection, we present a space-bounded *ftss* k-group consensus protocol CKP (Fig. 3) in complete networks under the C daemon. Note that we cannot obtain a space-bounded protocol on any $(n-k+1)$-connected network by combining the protocol CKP and the topology protocol [5], since the topology protocol is space-unbounded.

In CKP, when some processor p_i resets by detection of the inconsistency $|sus_i| > n-k$, the other processors reset synchronously. Synchronous reset means that each p_j $(\ne p_i)$ resets in the first step of itself after p_i resets, and, after that time on, exchanges messages only with reset processors. To implement this synchronous reset, CKP provides the following communication mechanism.

- **Detection of the duplicated read by the reader:** When p_j reads $R_{i,j}$, p_j can find whether p_i wrote $R_{i,j}$ after the last read of $R_{i,j}$.

- **Detection of the unread by the writer:** When p_i reads $R_{j,i}$, p_i can find whether p_j read $R_{i,j}$ after the last write to $R_{i,j}$.

These mechanism can be implemented if each of p_i and p_j can find which processor executed last in a step of itself. For this purpose, CKP uses a flag field F in each register. When p_i writes to $R_{i,j}$, p_i updates $R_{i,j}.F$ so as to be $last\_processor(p_i, p_j) = p_i$, where the function $last\_processor$ is defined as follows.

$$last\_processor(p_i, p_j)$$
$$= \begin{cases} p_i & \text{if } (id_i < id_j \wedge R_{i,j}.F \neq R_{j,i}.F) \\ & \quad \vee (id_i > id_j \wedge R_{i,j}.F = R_{j,i}.F) \\ p_j & \text{otherwise} \end{cases}$$

In Fig. 3, the following predicate is used.

$$first\_read(R_{j,i})$$
$$= (id_i < id_j \wedge sdata_{i,j}.F = rdata_{j,i}.F)$$
$$\vee\ (id_i > id_j \wedge sdata_{i,j}.F \neq rdata_{j,i}.F)$$

Each processor p_i decides whether p_i reads $R_{j,i}$ first after the last write to $R_{j,i}$ using this predicate.

Synchronous reset is implemented as follows. First, consider the case where some processor p_i detects the inconsistency $|sus_i| > n - k$, resets itself, and then requires all the other processors to reset themselves by setting a reset-request flag R to *true* in its each output register. Under the C daemon, the detection of the inconsistency and the set of reset-request flags are executed in a single atomic step. We call this atomic step a *reset-request step*. The processors p_i holds the reset-request flag in $R_{i,j}$ *true* until the processor p_j reads this request. On the other hand, p_j ($\neq p_i$) reads this request in the first step of itself after the reset-request step, and then resets itself.

First, we prove the communication mechanism.

Lemma 3: For any execution E of CKP under the C daemon, there exists some suffix of E in which, for any step a_i of any p_i, (1) there exists the last step a'_i of p_i before a_i in E and, (2) for any p_j ($\neq p_i$), $first\_read(R_{j,i})$ holds in a_i if and only if there exists a step of p_j between a'_i and a_i.

(Proof) Let E' be some suffix of E after every correct processor executes at least one step and all faulty processors have crashed. Consider any step of any p_i in E'. Since only correct processors execute steps in E', (1) there exists the last step a'_i of p_i before a_i in E.

Let p_j be any processor (maybe a faulty processor). In a'_i, p_i reads $R_{j,i}$ and stores the value to $rdata_{j,i}$. The processor p_i appends j to Rcv_i if and only if $first\_read(R_{j,i})$ holds. At the end of a'_i, p_i increments $sdata_{i,j}.F$ if j is in $R_{j,i}$, and then writes $sdata_{i,j}.F$ to $R_{i,j}$. At that time, $last\_processor(p_i, p_j) = p_i$ holds.

If p_j executes a step a_j between a'_i and a_i, at the end of a_j, $last\_processor(p_i, p_j) = p_j$ holds, and it continues to hold until a_i is executed. On the other hand, if p_j executes no step between a'_i and a_i, $last\_processor(p_i, p_j) = p_i$ continues to hold until a_i is executed.

In a_i, $sdata_{i,j}.F = R_{i,j}.F$ holds since only p_i writes to $R_{i,j}$ and $rdata_{j,i}.F = R_{j,i}.F$ holds since p_i first reads $R_{j,i}$ and sets $rdata_{j,i} = R_{j,i}$. Therefore, $last\_processor(p_i, p_j) = p_j$ holds if and only if $first\_read(R_{j,i})$ holds. This implies that (2) $first\_read(R_{i,j})$ holds in a_i if and only if there exists a step of p_j between a'_i and a_i. □

Next, we prove the synchronous reset.

Lemma 4: Any execution E of CKP under the C daemon has some suffix in which no processor executes a reset-request step.

(Proof) Consider some suffix E' satisfying Lemma 3. Let p_i be the first processor which executes a reset-request step a_i in E' if exists. In the reset step a_i, p_i writes the value *true* to each output register $R_{i,j}.R$. After a_i, p_i never changes the value of $R_{i,j}.R$ until p_j reads it. If a processor p_j ($\neq p_i$) executes its step after a_i, p_j reads the value *true* from $R_{i,j}.R$ in the first step a_j after that reset step, and then resets itself. In this a_j, p_j also reads all its input registers, therefore, p_j reads, in a_j at the latest, all messages written before the reset step. In a_j, p_j actually ignores the messages that p_j read from its input registers. In later steps, p_j ignores such ignored messages even if p_j reads them again. Therefore, after the reset step, every processor creates its total suspicion from the messages written after the reset-request step a_i and the suspected identifiers from its failure detecting process. This implies that, at the end of any step of any p_j ($\in P$) after the reset step, $sus_j \subseteq \bigcup_{p_i \in P, 0 \leq h} FP_i^{E,h}$ holds, therefore, $|sus_j| \leq n - k$ holds from the k-accuracy of the failure detector and p_j does not reset itself. That is, E has some suffix in which no processor executes a reset-request step. □

Now we show the correctness of CKP.

Theorem 3: If the failure detector is k-accurate and weakly complete, the protocol CKP is an $(n-k)$-*ftss* k-group consensus protocol in complete networks under the C daemon.

(Proof) We first show the convergence of variables sus_i. By Lemma 4, for any execution E of CKP under the C daemon, there exists some suffix E' in which no processor executes a reset-request step. In CKP, each processor p_i ignores any message if p_i has already read it. Since no processor executes a reset-request step in E', each p_i resets itself at most once when it first reads a reset request. Therefore, E' has some suffix $E'' = c''_0, c''_1, \cdots$ in which no processor resets itself. Now we show that Lemma 1 can be applied for E''. (1) Clearly, every $sus_i^{E'',h} \subseteq \{id_i | p_i \in P\}$

holds for any p_i and h. (2) $sus_i^{E'',h} \subseteq sus_i^{E'',h'}$ holds for any $p_i \in \mathcal{C}(E'')$ and any h and h' ($h \leq h'$). (3) Fairness of executions guarantees that for any p_i and p_j ($p_i, p_j \in \mathcal{C}(E'')$) and any h, there exists some h' such that $sus_i^{E'',h} \subseteq sus_j^{E'',h'}$. Finally, (4) for any sus_i ($p_i \in \mathcal{C}(E'')$) and any h ($h > 0$), $FP_i^{E'',h-1} \subseteq sus_i^{E'',h}$ holds.

By the above (1), (2), (3) and (4), and the facts of $\mathcal{C}(E'') = \mathcal{C}(E)$ and $\mathcal{F}(E'') = \mathcal{F}(E)$, there exist some set sus and some g_0 such that $\mathcal{F}(E) \subseteq sus$ and, for any $p_i \in \mathcal{C}(E)$ and g ($\geq g_0$), $sus_i^{E'',g} = sus$ hold. Since no processor executes a reset step in E'', $|sus| \leq n - k$ holds. Let *Active* be the value of $pick_k(\{id_1, id_2, \cdots id_n\} - sus)$. Then, $|Active| = k$ and $Active \subseteq \{id_j | p_j \in \mathcal{C}(E)\}$ hold. For any $p_i \in \mathcal{C}(E)$ and any $g \geq g_0$, $sus_i^{E'',g} = sus$ holds, and therefore, and $Active_i^{E'',g} = Active$ holds. Since E'' is a suffix of E, this implies that E is a legal execution for the k-group consensus problem. □

5. Conclusion

We considered fault-tolerant and self-stabilizing protocols using an unreliable failure detector. We defined k-accuracy of the failure detector, and showed the k-accuracy is necessary for *ftss* protocols for the k-group consensus problem. We also presented three $(n-k)$-*ftss* k-group consensus protocols using the k-accurate and weakly complete failure detector, (1) a space-unbounded protocol on complete networks under the R/W daemon, (2) a space-unbounded protocol on $(n-k+1)$-connected networks under the R/W daemon, and (3) a space-bounded protocol on complete networks under the C daemon. The first protocol shows that $(n-k)$-*ftss* k-group consensus can be solved in complete networks using a k-accurate and weakly complete failure detector even under the R/W daemon. We modified the protocol to the second one so that it should solve the problem in $(n-k+1)$-connected networks, a larger class of networks than the first one. However, these two protocols are space-unbounded. We resolved this disadvantages for the C daemon, which assumes larger atomic actions than the R/W daemon. Though the third protocol achieves space-bounded under the C daemon, it can be applied only to complete networks. The space-boundedness is an important requirement for self-stabilizing protocols. Practically, we can prepare sufficiently large spaces for unbounded variables if they are used in some non-self-stabilizing protocol. However, self-stabilizing protocols and *ftss* protocols can be started from any configuration, and therefore, we cannot prepare a sufficiently large space for unbounded variables in advance. It is one of our future works to investigate the possibility of a space-bounded *ftss* k-group consensus protocol under the R/W daemon.

Acknowledgment

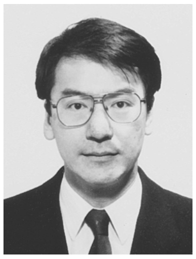
This work was supported in part by the Ministry of Education, Science, Sports and Culture, Japan under the Grant-in-Aid for Scientific Research on Priority Areas B(2) (No.10205218).

References

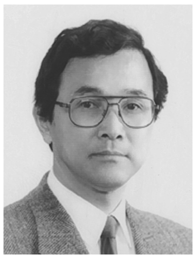
[1] A. Gopal and K.J. Perry, "Unifying self-stabilization and fault-tolerance," Proc. 12th ACM Symposium on Principles of Distributed Computing, pp.195–206, 1993.
[2] T. Chandra, V. Hadzilacos, and S. Toueg, "The weakest failure detector for solving consensus," J. Assoc. Comput. Mach., vol.43, no.4, pp.685–722, 1996.
[3] T. Chandara and S. Toueg, "Unreliable failure detectors for reliable distributed systems," J. Assoc. Comput. Mach., vol.43, no.2, pp.225–267, 1996.
[4] E. Anagnostou and V. Hadzilacos, "Tolerating transient and permanent failures," Proc. 7th International Workshop on Distributed Algorithms (LNCS 725), pp.174–188, 1993.
[5] T. Masuzawa, "A fault-tolerant and self-stabilizing protocol for topology problem," Proc. 2nd Workshop on Self-Stabilizing Systems, pp1.1–1.15, 1995.
[6] J. Beauquier and S. Kekkonen-Moneta, "On FTSS-solvable distributed problems," Proc. 3rd Workshop on Self-stabilizing Systems, pp.64–79, 1997.
[7] S. Katz and K.J. Perry, "Self-stabilizing extensions for message-passing systems," Proc. 10th ACM Symposium on Principles of Distributed Computing, pp.91–101, 1990.
[8] B. Awerbuch, B. Patt-Shamir, and G. Varghese, "Self-stabilization by local checking and correction," Proc. 32nd IEEE Symposium on Foundations of Computer Science, pp.268–277, 1991.
[9] B. Awerbuch, B. Patt-Shamir, G. Varghese, and S. Dolev, "Self-stabilization by local checking and global reset," Proc. 8th International Workshop on Distributed Algorithms, pp.326–339, 1994.
[10] T. Masuzawa and Y. Katayama, "On self-stabilizing algorithms," J. Inf. Processing Society of Japan, vol.34, no.11, pp.1358–1365, 1993.
[11] S.-T. Huang, "Leader election in uniform rings," ACM TOPLAS, vol.15, no.3, pp.563–573, 1993.
[12] Y. Katayama, T. Masuzawa, and N. Tokura, "Self-stabilizing ring orientation algorithm under the C-daemon," IEICE Trans., vol.J77-D-I, no.12, pp.777–784, Dec. 1994.
[13] R. Guerraoui and A. Schiper, "Γ-accurate failure detectors," Proc. 10th International Workshop on Distributed Algorithms, pp.269–286, 1996.
[14] S. Dolev, K. Israeli, and S. Moran, "Self-stabilization of dynamic systems assuming only read/write atomicity," Proc. 9th ACM Symposium on Principles of Distributed Computing, pp.103–117, 1990.
[15] J. Burns, M. Gouda, and R.E. Miller, "Stabilization and pseudo-stabilization," Distributed Computing, vol.7, no.1, pp.35–42, 1993.

Hiroyoshi Matsui received the B.E. degree from Waseda University in 1994, and the M.E. degree from Nara Institute of Science and Technology (NAIST) in 1996. He joined DDI Pocket in 1996. His research interests include distributed algorithms.

Michiko Inoue received her B.E., M.E., and Ph.D. degrees in computer science from Osaka University in 1987, 1989, and 1995 respectively. She is an instructor of Graduate School of Information Science, Nara Institute of Science and Technology (NAIST). Her research interests include distributed algorithms, parallel algorithms, and design and test of digital systems. She is a member of IEEE, IPSJ, and JSAI.



Toshimitsu Masuzawa received the B.E., M.E. and D.E. degrees in computer science from Osaka University in 1982, 1984 and 1987. He had worked at Education Center for Information Processing, Osaka University between 1987–1990, and had worked at Faculty of Engineering Science, Osaka University between 1990–1994. He is now an associate professor of Graduate School of Information Science, Nara Institute of Science and Technology (NAIST). He was also a visiting associate professor of Department of Computer Science, Cornell University between 1993–1994. His research interests include distributed algorithms, parallel algorithms and graph theory. He is a member of ACM, IEEE, EATCS and the Information Processing Society of Japan.



Hideo Fujiwara received the B.E., M.E., and Ph.D. degrees in electronic engineering from Osaka University, Osaka, Japan, in 1969, 1971, and 1974, respectively. He was with Osaka University from 1974 to 1985 and Meiji University from 1985 to 1993, and joined Nara Institute of Science and Technology in 1993. In 1981 he was a Visiting Research Assistant Professor at the University of Waterloo, and in 1984 he was a Visiting Associate Professor at McGill University, Canada. Presently he is a Professor at the Graduate School of Information Science, Nara Institute of Science and Technology, Nara, Japan. His research interests are logic design, digital systems design and test, VLSI CAD and fault tolerant computing, including high-level/logic synthesis for testability, test synthesis, design for testability, built-in self-test, test pattern generation, parallel processing, and computational complexity. He is the author of Logic Testing and Design for Testability (MIT Press, 1985). He received the IECE Young Engineer Award in 1977, IEEE Computer Society Certificate of Appreciation Award in 1991, Okawa Prize for Publication in 1994, and IEEE Computer Society Meritorious Service Award in 1996. He is an advisory member of IEICE Trans. on Information and Systems and an editor of IEEE Trans. on Computers, J. Electronic Testing, J. Circuits, Systems and Computers, J. VLSI Design and others. Dr. Fujiwara is a fellow of the IEEE and a Golden Core member of the IEEE Computer Society as well as a member of the Institute of Electronics, Information and Communication Engineers of Japan and the Information Processing Society of Japan.