ELSEVIER Journal of Information Sciences 108 (1998) 157–180

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

Reaching strong consensus in the presence of mixed failure types

Hin-Sing Siu [a,*], Yeh-Hao Chin [b], Wei-Pang Yang [c]

[a] *Department of Industrial Engineering and Management, Mingchi Institute of Technology, Taipei, Taiwan 24306, ROC*

[b] *Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 30043, ROC*

[c] *Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan 30050, ROC*

Received 10 October 1995; received in revised form 24 May 1997; accepted 26 August 1997

Abstract

The *Strong Consensus* (SC) is a variant of the conventional distributed consensus problem. The protocol designed for the SC problem requires that the agreed value among fault-free processors be one of the fault-free processor's initial value. The SC problem is re-examined with the assumption of *mixed failure types* (also referred to as the *hybrid fault model*). Compared with the features of the existing protocols, the underlying network topologies of the proposed protocol do not have to be fully connected, the mixed failure types can be tolerated, and no prior information of the system's faulty status is required. The proposed protocol can tolerate a maximum number of faults to enable each fault-free processor to reach an agreement with a minimum number of message exchanges.

Keywords: Byzantine agreement; Distributed consensus; Fault-tolerant distributed system; Mixed failure types; Nonfully connected network; Strong consensus

* Corresponding author. E-mail: hssiu@ccsun.mit.edu.tw. fax: 886 2 29041914.

0020-0255/98/$19.00
PII: S0020-0255(97)10054-8

1. Introduction

The *Distributed Consensus* problem [1–5] is one of the most important problems in designing a fault-tolerant distributed system. A variant of distributed consensus, called the *Strong Consensus* (SC) problem, was introduced by Neiger [6]. In the SC problem, each processor A starts with an initial value, belongs to a finite set, V, of all possible values ($|V| = m$). The protocol for the SC problem is to enable all fault-free processors to obtain a common value. After the execution of the protocol, the common value obtained by the fault-free processors shall be the value that satisfies the following conditions:
1. *Agreement*: All fault-free processors agree on the same common value v, and
2. *strong validity*: v is the initial value of some fault-free processors.

In practice, most network topologies are not fully connected and network processors may be subjected to different types of failure simultaneously [1]. From the standpoint of disruptive effects, the processor failure types can be divided into two disjoint subsets: *dormant faults* and *arbitrary faults*, termed by Meyer and Pradhan [7]. A dormant fault is defined as a fault that does not contaminate the content of a message and produces missing values detectable by all fault-free processors. An arbitrary fault refers to the case when the behavior of a fault is not restricted.

However, the SC protocol proposed by Neiger [6], like most conventional consensus protocols [8,9,3], is designed to handle arbitrary faulty processors in a fully connected network. Based on the discussion of [10,7,11], the protocol of [6] cannot tolerate a maximum number of faults when dormant faults are considered because the faulty behaviors of the dormant faults are more consistent than that of the arbitrary faults. Also, the protocol cannot be applied to a nonfully connected network such as the one shown in Fig. 1.

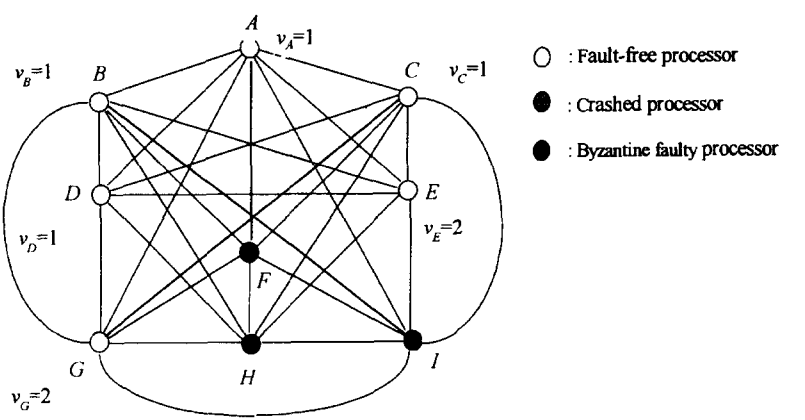


Fig. 1. A network with mixed failure types.

Some existing protocols [10,7,11] are designed to solve the consensus problem with *mixed failure types* (also referred to as the *hybrid fault model*). However, they will not guarantee the strong validity condition as specified by [6]. Moreover, the major limitation of these protocols is that the number of arbitrary faulty processors must be known prior to execution of the protocols. However, this requirement violates the general assumption of the consensus problem – that a fault-free processor does not know which processor is faulty [3,5]. Furthermore, Shin and Ramanathan [12] found that it is impractical to run diagnostics to detect all arbitrary faults in a network. Thus, these protocols cannot solve the SC problem with mixed failure types.

The strategies used in previous research of the consensus problem can be classified into two groups: *deterministic strategy* and *nondeterministic strategy* [13,14]. Each step for executing deterministic strategy is predetermined, such as all processors execute the same protocol, and all processors start and stop the execution of protocol at the same time. Conversely, certain steps of a protocol are not predefined in the nondeterministic strategy. For example, a processor can stop the executing of the protocol *early* when it can decide the common value. However, in such a strategy, a processor cannot decide whether another processor has stopped early or crashed when no message was received from it; thus, the nondeterministic strategy is inappropriate for mixed failure types. Hence, the research in the consensus problem with mixed failure types (including this paper) concentrates on the deterministic strategy.

We re-examined the SC problem with mixed failure types in a *synchronous* network [1] and proposed a protocol that can solve the problem under the following assumptions:

1. The underlying network may not be fully connected and the communication links in the network are assumed to be fault-free.
2. Let n be the total number of processors. Each processor's identifier is unique. A processor does not know the fault status of another processor.
3. Let c be the *connectivity* of the underlying network. Due to the Menger theorem [15], at least c disjoint paths exist between any pair of processors S and R if the connectivity of the network is c. For any two paths the only common components are S and R.
4. Let P_a be the number of processors subjected to arbitrary faults.
5. Let P_d be the number of processors subjected to dormant faults.
6. Let V be the set of all possible values of the SC problem, m be the size of V, and each processor uses a predefined enumeration of the values in V: $v_1, v_2, \ldots, v_m$.

[1] In a *synchronous* network, the bounds for processing and communication delay in fault-free components (processors and links) are finite and are known by all fault-free processors [13].

The protocol designed for solving the SC problem with mixed failure types is called SCMIX. SCMIX enables all fault-free processors to obtain a common value for solving the SC if $n > \max\{mP_a + P_d, 3P_a + P_d\}$ and $c > 2P_a + P_d$. It uses a minimum number of message exchanges and can tolerate a maximum allowable number of faulty processors. SCMIX is based on the *oral message model* (one of the deterministic strategy) [3], which has two phases: *the message exchange phase* and *the decision making phase*. The goal of the message exchange phase is to collect the messages and that of the decision making phase is to compute a common value for solving the SC problem. The formal descriptions of these phases are given in Section 3.

The remainder of the paper proceeds as follows. In Section 2 we give the conditions for the SC problems. Section 3 proposes the detail descriptions of the proposed protocol. The analysis and evaluation of SCMIX are presented in Section 4. The conclusion is given in Section 5.

2. The conditions for strong consensus

In order to solve the SC problem, the number of message exchange rounds required and the number of faulty processors allowed shall first be considered.

2.1. The number of rounds required by SCMIX

Since a processor does not know the fault status of another processor, each fault-free processor requires $t + 1$ rounds [2] to exchange the messages needed to reach an agreement [3,6,5], where $t = \lfloor (n-1)/(\max\{m, 3\}) \rfloor$. Fischer and Lynch [16] also pointed out that the $t + 1$ rounds are the minimum number of rounds required to reach a common value when the network's fault status is unknown. Therefore, the minimum number of rounds required by SCMIX is $t + 1$.

2.2. The number of allowable faulty processors by SCMIX

Essentially, the fault tolerant capabilities of a network depend on the total number of processors and the network topology (*connectivity*). For example, every faulty processor can prevent the fault-free processors from achieving an agreement if the network topology is a tree. To generalize, the complete characterizations of constraints on failures for the SC problem are shown in Theorem 1. The SC protocol that meets Theorem 1 is given in Section 3.

[2] A round denotes the interval of message exchange between a pair of processors [7,14].

Theorem 1. *For any network with n processors and c connectivity, SC can be achieved if*:

(1) $n > \max\{mP_a + P_d, 3P_a + P_d\}$, *and*

(2) $c > 2P_a + P_d$

Proof. The first constraint specifies the number of processors required and follows the concept of [3,7,17]. Via a time-out mechanism (the approach is presented in Section 3), a fault-free processor can detect the occurrence of a dormant fault in a processor and ignore all messages from that processor if no message is received from it within a predefined time interval. Then the network behaves as a network with $n - P_d$ processors. The constraint $n - P_d > 3P_a$, namely $n > 3P_a + P_d$, requires that the SC protocol also solve the consensus problem [7,17]. Follow the results of [7], the constraint $n - P_d > mP_a$, namely $n > mP_a + P_d$, is required to solve the SC problem. Thus, $n > \max\{mP_a + P_d, 3P_a + P_d\}$ is the constraint on the number of processors required.

On the other hand, the second constraint specifies the connectivity required. In each round, every processor sends its message to other processors. In order to decide whether a processor has sent out its message, by the concept of *majority*, the total number of arbitrary faulty processors must be less than half of $c - P_d$, namely $c > 2P_a + P_d$ [9]. Otherwise, such a goal cannot be reached. □

By Theorem 1, the maximum number of tolerable faulty processors by SCMIX is $P_a + P_d$ if $n > \max\{mP_a + P_d, 3P_a + P_d\}$ and $c > 2P_a + P_d$ (the formal proof is shown in Theorem 7).

3. Basic concept and approaches

In a nonfully connected network, a processor can act either as *sender, receiver*, or *relay* depending on the message flow. A message, sent from a sender to a receiver, may be passed through some relay. The message may be influenced by either the sender (dormant or arbitrary fault), some faulty relay, or both. To solve the SC problem, SCMIX must therefore completely remove the influences of dormant faulty senders, arbitrary faulty senders, and faulty relays.

In order to solve the SC problem, based on the oral message model, each processor should execute t message exchange rounds (stated in the Section 2) to collect the messages. At each round, a processor should broadcast its message to all processors and receive messages from all processors. To remove the influence caused by the faulty relays, each processor uses the *fault-tolerant virtual channel* (FTVC) protocol to broadcast its message. The formal description is presented in Section 3.1. Using FTVC, a fault-free processor can therefore detect that a processor Q is faulty if no message is received from Q during

one or some rounds. Once a processor Q is detected as faulty, the messages received from Q can be ignored at each subsequent round. Such an approach is called the *Absent Rule* AR_{SC} and its formal definition is presented in Section 3.2. The absent rule can handle the faults, especially the dormant faults, that do not send messages as it should be. In [17], we found that the traditional majority vote is inappropriate for mixed failure types (also see Section 3.3). As a result, a new voting scheme $VOTE_{SC}$ is proposed for solving the SC problem with mixed failure types, and the formal description of $VOTE_{SC}$ is presented in Section 3.3. Since the messages received from the faulty processors, detected by the absent rule, are ignored, the network behaves like a network with $n - P_d$ processors. Based on the majority concept, the total number of the tolerable faulty processors by SCMIX can be increased. With FTVC protocol, the absent rule AR_{SC} and voting function $VOTE_{SC}$, the proposed protoocol SCMIX consists of: *the message-exchange phase* and *the decision-making phase*. The main functions of these phases are shown in Fig. 2. Using these two phases, SCMIX can solve the SC problem.

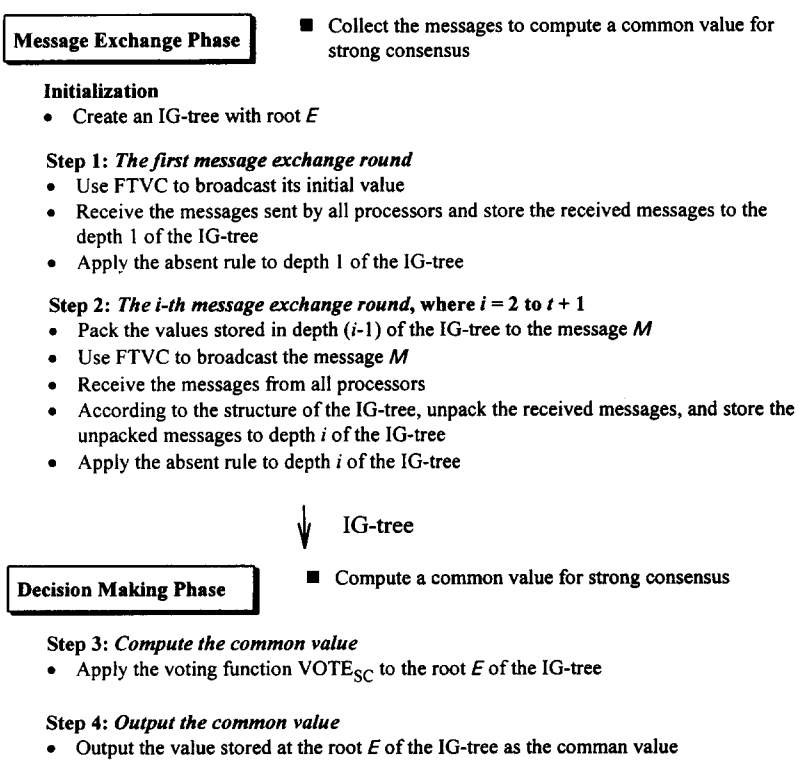


Fig. 2. The basic approaches of SCMIX.

As for the data structure used to collect the messages, each fault-free processor maintains a tree structure, called the *information gathering tree* (IG-tree) [8], of depth $t + 1$. It is organized and labeled as follows. Each vertex of the IG-tree is labeled by a non-repeating sequence, α, of processor identifiers. The root of the IG-tree is labeled by the empty sequence **E** and the parent of a vertex labeled by sequence αp is labeled α. Because all sequences are nonrepeating, the root of the IG-tree has n children, each child of the root has $n - 1$ children, and the vertices at depth t each has $n - t$ leaves as it children. If a vertex is labeled αp, it is said to correspond to processor p. The root of the IG-tree corresponds to no processor. Note that each level of an IG-tree contains a round of received messages. No repeating processor identifier can avoid the recursive influences made by a faulty processor.

SCMIX is illustrated by an example that shows the complete procedure for executing SCMIX on the fault-free processor A in the network shown in Fig. 1. The same procedure is executed by each fault-free processor. Suppose that the dormant faulty processor F does not send any messages during the entire execution of the protocol and $m = 3$. When SCMIX is finished, all the fault-free processors reach a common value '1' that is the initial value of processor A; i.e., the *Agreement* and *Strong Validity* condition of the SC problem are both satisfied. Hence, SCMIX does solve the SC problem with mixed failure types. The procedure of SCMIX on processor A is presented as follows.

Message exchange phase

Initialization

- Create an IG-tree with root **E**

  In the initial step, processor A creates an IG-tree with root **E**.

Step 1: The first message exchange round

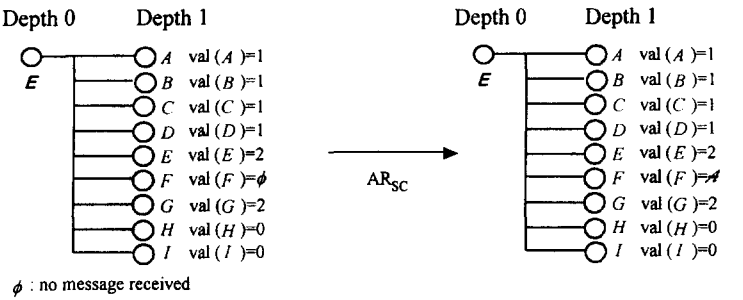
- Use FTVC to broadcast its initial value

  Processor A uses FTVC to broadcast its initial value to all processors.
- Receive the messages sent by all processors and store the received messages to the depth 1 of the IG-tree

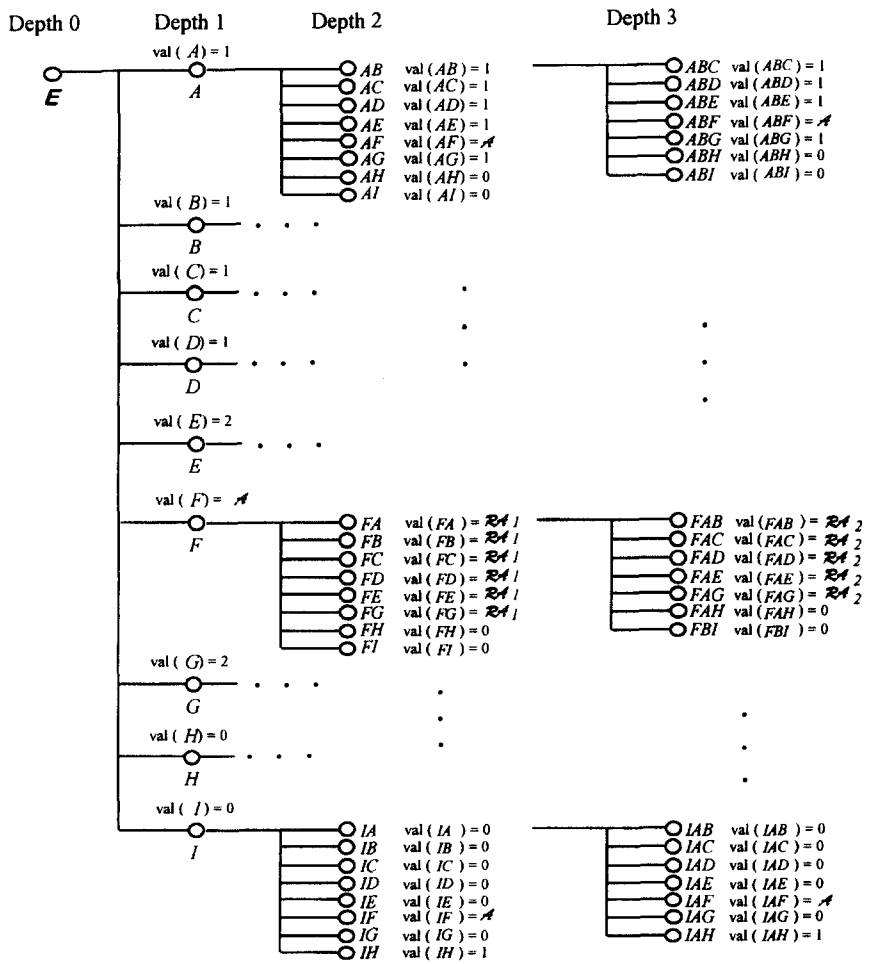
Although the message passing may be influenced by the faulty processors F, H, and I, processor A receives the messages sent by all processors, and stores the messages to depth 1 of its IG-tree as shown in Fig. 3(a). That means FTVC can remove the influences of the faulty relay processors. For example, A stores the value '1' received from processor B in the vertex B, denoted as $\mathrm{val}(B) = 1$, at depth 1 of IG-tree as shown in Fig. 3(a).

- Apply the absent rule $\mathrm{AR_{SC}}$ to depth 1 of the IG-tree

To remove the influence of a dormant faulty processor, processor A then applies AR_{SC} to depth 1 of its IG-tree as shown in Fig. 3(b). Note that the value stored as vertex F in the IG-tree is $\varnothing$ as shown in Fig. 3(a). That means processor A does not receive any messages from F and uses the value $\varnothing$ to represent the message from F. After the absent rule is applied, processor A will use the value $\mathscr{A}$ to replace the messages received from F as shown in Fig. 3(b). The

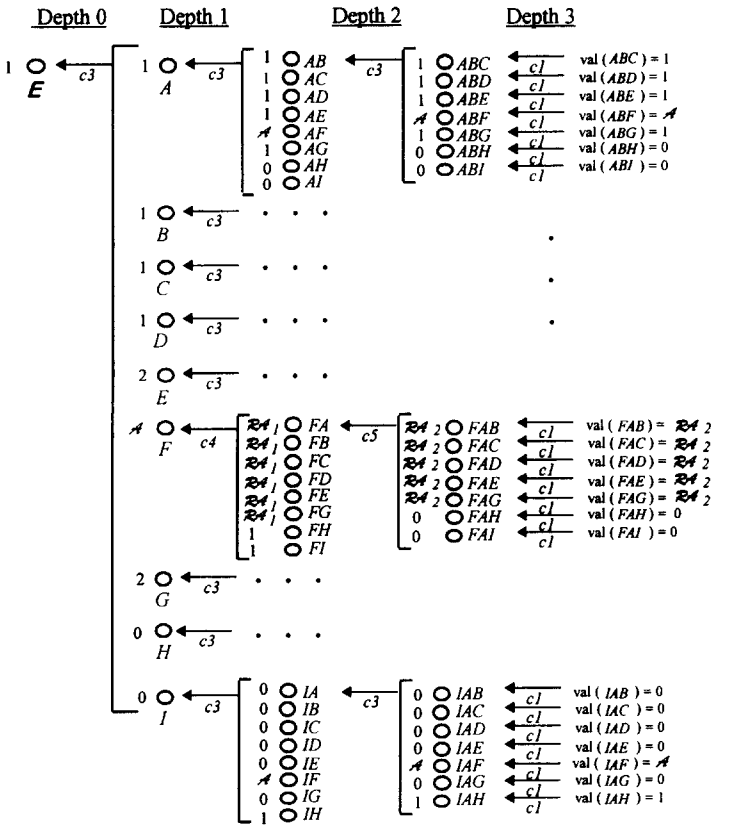


(a) (b)



(c)

Fig. 3. SCMIX solves the SC problem for processor A in the network model shown in Fig. 1: (a) The first round IG-tree. (b) After the absent rule is applied. (c) After the message exchange phase (d) Applying the VOTE_{SC}.



(d)

Fig. 3 (*continued*)

value $\mathscr{A}$ will be relayed to all receivers as value $\mathscr{RA}_1$ and the value $\mathscr{RA}_j$ will be relayed to all receivers as value $\mathscr{RA}_{j+1}$ (the meaning of value $\mathscr{A}$ and $\mathscr{RA}_j$ will be described later), where $1 \leqslant j \leqslant t$.

Step 2: the second message exchange round

- Pack the values stored in depth 1 of the IG-tree to the message M

  Processor A packs the values stored in depth 1 of the IG-tree to the message M, namely (1,1,1,1,2, $\mathscr{RA}_1$, 2,0,0). Note that the value $\mathscr{A}$ is replaced by $\mathscr{RA}_1$ by using the absent rule.
- Use FTVC to broadcast the message M

  Using FTVC, processor A broadcasts the massage M to all processors.
- According to the structure of the IG-tree, unpack the received messages, and store the unpacked messages to depth 2 of the IG-tree.

- Processor A receives the messages sent by all processors, unpacks these messages, and stores the unpacked messages to the vertices at depth 2 of its IG-tree.
- Apply the absent rule to depth 2 of the IG-tree.

To remove the influence of the detected faulty processors, processor A applies the absent rule to depth 2 of its IG-tree.

In the third round, processor A executes the same procedure as in the second round did. Processor A broadcast, using FTVC, the messages stored at depth 2 of its IG-tree, receives the messages sent by all processors, and stores the received messages to depth 3 of its IG-tree. After the message exchange phase, the messages collected in A's IG-tree is presented in Fig. 3(c) (due to space limitations, only a part of the IG-tree is shown). For example, the val(ABC) (val(ABC) = 1) represents that the message was first sent by processor A to processor B, and then processor B relayed this message to processor C. Finally, A received this message from C and stored it in vertex ABC as shown in depth 3 of the IG-tree in Fig. 3(c). In this example, 505 ($1 + 9 * 8 * 7$) vertices are created in the IG-tree.

Decision making phase

Step 3: Compute the common value

- Apply the voting function VOTE_{SC} to the root **E** of the IG-tree.
- Processor A applies the voting function VOTE_{SC} to its IG-tree to compute the common value for strong consensus. Note the value $\mathscr{A}$ (excluding the last round) is not counted at the time the VOTE_{SC} is taken. The value '1' is stored in the root **E** after VOTE_{SC} is applied as shown in Fig. 3(d).
- Output the value stored at the root **E** of the IG-tree as the common value.

Finally, processor A selects the value '1' stored in the root of the IG-tree as the common value.

The detailed descriptions of the above steps for removing the influences of the multiple faulty processors are presented below.

3.1. Removing the influence of a faulty relay

To remove the influence of a faulty relay, a protocol, called FTVC, provides a *fault-tolerant virtual channel* on the physical links in a nonfully connected network. To illustrate the concept of FTVC, we first consider the case of a single sender S and a single receiver R. S uses FTVC to send its message m_s to R. Analyzing this exchange will enable us to portray the general situation in which every sender sends a message to every receiver. For example, when FTVC is applied to the network model shown in Fig. 4(a), receiver R can receive the fault-free message sent by sender S; while in the case of Fig. 4(b), R can positively detect that S did not send a message to it even if it does receive the false message sent by the arbitrary faulty relay.

As the Menger theorem [15] states, at least c disjoint paths exist between S and R if the connectivity of the network is c. Hence S is able to send c copies of its messages through c disjoint paths to R. The c disjoint paths between S and R

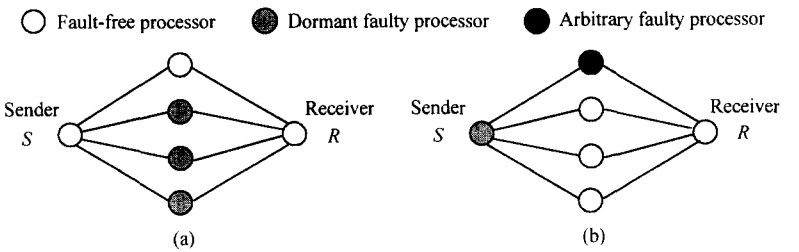


Fig. 4. An example of the function of FTVC.

can be predefined as stated in [9,7], and the path information is distributed to the relaying processors between S and R. A detailed description of the path information distribution is presented in Appendix A. According to the path information, a relay processor receives the message (R, S, m_s) from a predefined immediate predecessor and sends the message to a predefined immediate successor. Since the network is synchronous, the predefined immediate successor P of S should have the message sent by S after a predefined time interval [13]; otherwise, it knows that S is faulty. When P receives no message from S, it will relay the symbol $\varnothing (\varnothing \notin V)$ to its immediate successor along the predefined disjoint path between S and R to reflect the faulty status. These are the concepts of the *transfer rules* obeyed by each relay processor. The formal definition of the transfer rules is presented in Appendix B.

According to the transfer rules, an arbitrary faulty relay can modify at most one message, and a dormant faulty relay can drop at most one message. In the worst case, R will receive $c - P_d$ copies of messages sent by S. Applying the majority vote MAJ to these messages, R can determine what message was sent by S if the constraint on connectivity, namely $c > 2P_a + P_d$, holds. MAJ has three possible outcomes:

Case 1: m_s, if S is fault-free.

Case 2: $\varnothing$, if S does not send the message to R.

Case 3: *Arbitrary value*, if S has an arbitrary fault.

In case 1, R receives the message m_s sent by the fault-free sender S when MAJ is applied to the receiver messages. If S does not send the message to R (case 2), R will use $\varnothing$ as the message sent by S because the *major* of $c - P_d$ copies of messages is $\varnothing$. The third outcome of MAJ implies that the received message is not only contaminated by a faulty relay, but is also contaminated by an arbitrary faulty sender. FTVC is unable to remove the influence in such a case; hence such an outcome for MAJ shall be an arbitrary value.

3.2. Removing the influence of a dormant faulty sender

Each fault-free sender must send its messages to all receivers in each round of the message exchange phase. As mentioned in Section 3.1, a receiver can

therefore detect that a sender is faulty if no message is received from the sender (the output of MAJ of FTVC is $\varnothing$). A fault-free receiver R can detect that a sender S is faulty if no message is received from S. If R receives no message from S at the rth round, all messages received from S (directly) at the rth round and any subsequent rounds will be replaced by the value $\mathscr{A}$; and this value will be relayed to the other receivers as value $\mathscr{R}\mathscr{A}_1$. In each subsequent round, the value $\mathscr{R}\mathscr{A}_j$ will be relayed to the other receivers as value $\mathscr{R}\mathscr{A}_{j+1}$ ($\mathscr{A}$ and $\mathscr{R}\mathscr{A}_j \notin V$), where $1 \leqslant j \leqslant t$.

Semantically, the value $\mathscr{A}$ is represented as an *absentee vote*, and sender S is treated as an *absentee*. Hence, the voting ticket of S is ignored during the decision making phase. The value $\mathscr{R}\mathscr{A}_j$ will be interpreted as *the j*th *time an absent vote is reported.* Receiver R will report to all other receivers that S is an absentee, and then the faulty sender S will be forced out of the game of agreement; thus, S has no influence on the others when the voting function $\mathrm{VOTE}_{\mathrm{SC}}$ is taken in the decision making phase. The approach is called the Absent Rule ($\mathrm{AR}_{\mathrm{SC}}$) and it can be formalized as follows.

> "$\mathrm{AR}_{\mathrm{SC}}$: When receiver R receives no message directly from sender S in the rth round, then all messages received from S in the rth and any subsequent rounds will be replaced by value $\mathscr{A}$, and this value will be relayed to the other receivers (if any) as value $\mathscr{R}\mathscr{A}_1$. In each subsequent round, the value $\mathscr{R}\mathscr{A}_j$ will be relayed to the other receivers as value $\mathscr{R}\mathscr{A}_{j+1}$, where $1 \leqslant j \leqslant t$."

3.3. Removing the influence of an arbitrary faulty sender

After the message exchange phase, the messages collected in a fault-free receiver's IG-tree are free from the influence of faulty relays and the dormant faulty senders. However, the messages may still be contaminated by arbitrary faulty senders. In order to reach an agreement, such influences must be removed in the decision making phase.

Conventionally, the influence of arbitrary faulty senders is removed by means of a recursive majority vote when only arbitrary faults are considered [3,6]. The main concept used in these protocols is *majority* because a majority of the processors in the network are assumed to be fault-free. However, this concept is inappropriate for mixed failure types because a majority of the processors *may* also fail. Using Fig. 5 as an example, there are four faulty processors (three dormant faulty processors and one arbitrary faulty processor; i.e., $P_{\mathrm{d}} = 3$ and $P_{\mathrm{a}} = 1$), a greater number than the number of fault-free processors (three). Suppose that $m = 3$ (the number of values of V). The bound on the constraints on failures, namely $n > \max\{mP_{\mathrm{a}} + P_{\mathrm{d}}, 3P_{\mathrm{a}} + P_{\mathrm{d}}\}$, holds because $7 > 3 * 1 + 3$. However, the fault-free processors, A, B and C, are unable to

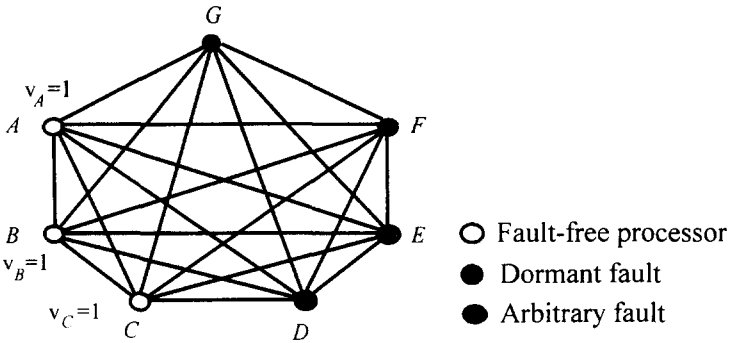


Fig. 5. A fully connected network with mixed failure types ($n = 7$).

reach agreement when a conventional majority vote is taken. A detailed description of this example is presented in Appendix C. Thus, a new voting scheme VOTE_{SC} should be proposed to solve the SC problem with mixed failure types.

By the constraint on the number of processors required, namely $n > \max\{mP_a + P_d, 3P_a + P_d\}$, it can tolerate $k(= \max\{m, 3\})$ more dormant faulty senders because $k(P_a - 1) + (P_d + k) = kP_a + P_d$, where $P_a \geqslant 1$ if the network eliminates one arbitrary faulty sender. This phenomenon can be used by VOTE_{SC} to remove the influence of an arbitrary faulty sender. The basic concept of VOTE_{SC} is as follows. Let P be a fault-free processor and σ be the vertex at depth i of P's IG-tree, $1 \leqslant i \leqslant t$. If P detects that $k(t - i + 1) + [(n - 1) \bmod k]$ children of σ have value $\mathscr{A}$, it uses the original value stored at σ, namely $\text{val}(\sigma)$, as the output of VOTE_{SC} to remove the influence of the arbitrary faulty sender as in the above discussion; otherwise, it uses the most common value of children of σ as the output of VOTE_{SC}.

VOTE_{SC} is always correct if vertex σ corresponds to a fault-free or a dormant faulty sender since each fault-free receiver has the same message sent by the sender. On the other hand, the output of VOTE_{SC} *may* be contaminated by Q after our approach is applied if vertex σ corresponds to an arbitrary faulty sender Q (Q cooperates with other arbitrary faulty senders to prevent the fault-free processors from achieving a common value). However, the influence of Q can still be removed during upper level voting if $n > \max\{mP_a + P_d, 3P_a + P_d\}$. Appendix D presents the formal definition of VOTE_{SC}.

4. Analysis and evaluation

SCMIX removes the influence of processors subjected to various types of failures to enable all fault-free processors to reach a common value to solve

the SC problem. Since a fault-free processor cannot know the fault status of the other processors, $t+1$ rounds are required to reach an agreement. SCMIX uses the approaches stated in Section 3 to remove the multiple faulty components, and these approaches can be presented as the following primitives:

- FTVC_SEND(m, Q): send the message m to processor Q by using FTVC.
- FTVC_RECEIVE(m, Q): receive the message m from processor Q by using FTVC.
- ABSENT_RULE(r): apply the absent rule $\mathrm{AR}_{\mathrm{SC}}$ to depth r of the IG-tree.
- $\mathrm{VOTE}_{\mathrm{SC}}(s)$: apply the function VOTE to vertex s.

Some additional primitives should be presented to ensure a thorough solution:

- CREATE$(\sigma Q, v)$: create the vertex σQ, and set $\mathrm{val}(\sigma Q) = v$.
- PACK(r, m): fold depth r of the IG-tree to the message m.
- UNPACK(m, r): according the structure of depth r of the IG-tree, unfold the message.
- OUTPUT(v): output the value v.

Using the above primitives, the formal procedure of SCMIX is stated as follows.

```
1. Protocol SCMIX (for each processor P)
2. begin
3. /* Initialization */
4.      CREATE (E, NULL);
5. /* Message Exchange Phase */
6. /* The first round */
7.      for Q ∈ N do
8.        FTVC_SEND(v_p, Q);
9.      for Q ∈ N do
10.     begin
11.           FTVC_RECEIVE(v_q, Q); /* v_q is the initial value of processor Q
                */
12.           CREATE(Q, v_q);
13.     end;
14.     ABSENT_RULE(1);
15.     /* round 2 to round t + 1 */
16.     for r = 2 to t + 1 do
17.     begin
18.       PACK(r − 1, m);
19.       for Q ∈ N do
20.           FTVC_SEND(m, Q);
21.       for Q ∈ N do
22.       begin
23.           FTVC_RECEIVE(m, Q);
24.           UNPACK(m, r);
```

```
25.            for σ ∈ m do
26.                begin
27.                  v = val(σ);
28.                  CREATE(σQ, v)
29.                end
30.          end;
31.          ABSENT_RULE(r)
32.        end;
33. /* Decision Making Phase */
34.        OUTPUT(VOTE_SC(E))
35. end.
```

4.1. Correctness

The goal of SCMIX is to enable all fault-free processors to reach a common value to solve the SC problem; thus, the correctness of SCMIX can be proven from the fact that the common value of each fault-free processor satisfies the conditions of Agreement and Strong Validity. To reach a common agreement, each fault-free processor must be insulated from contamination by faulty processors. As stated in Section 3, SCMIX uses FTVC to remove the influences of faulty relays during each round of message exchange. To remove the influence of dormant faulty senders, SCMIX applies AR_{SC} to be received messages after each round. Finally, it applies $VOTE_{SC}$ to the messages, received in the message exchange phase, to remove the influence of arbitrary faulty senders. When all contamination by faulty processors has been removed, an agreement is reached. This is the basic concept for proving the correctness of SCMIX.

Since SCMIX uses the IG-tree (based on the oral message model) to collect the messages as presented in Section 3, some concepts and terminology used by [8] are presented here. A vertex σ is called *common*, if each fault-free processor computes a same value for σ. In other words, a common value for solving the SC problem can be reached if the root of each fault-free processor's IG-tree is common. To prove the root is common, the term *common frontier* is defined as follows. If every root-to-leaf path in an IG-tree contains a common vertex, then the collection of common vertices forms a common frontier. By the *frontier lemma* in [8], the fault-free processor's IG-tree root is common if the common frontier exists on each fault-free processor's IG-tree. Hence, a common agreement can be reached among the fault-free processors if a common frontier does exist in each fault-free processor's IG-tree.

To prove the correctness of FTVC, the output of MAJ shall be proven free from the influence of faulty relays. Thus, we shall prove that a fault-free receiver can receive a message sent by a fault-free sender, or can detect that the sender did not send a message to it. Accordingly, we first define the *consistent vertex* as follows.

Consistent vertex. vertex $\alpha(\alpha = \sigma i)$ in a fault-free receiver's IG-tree is a consistent vertex if sender i is fault-free or dormant fault. By the behavior of i, all fault-free receivers receive the identical message sent by i. Although a processor does not know which vertex is consistent, the consistent vertices do exist since some processors in the network are fault-free or dormant fault.

According to the definition of a consistent vertex, all fault-free receivers should receive an identical message sent by a sender if the influence of a faulty relay is removed. Therefore, the consistent vertices of an IG-tree are common. Since the maximum number of arbitrary faulty processors is $P_a(\leqslant t)$, each root-to-leaf path has at least one consistent vertex. Therefore, the common frontier does exist in the IG-tree. Thus the root is a common vertex due to the existence of a common frontier. A common agreement is reached among all fault-free processors; thus, the SC problem with mixed failure types is solved.

To summarize the semantics for the following lemmas and theorems, Lemma 1 indicates that a fault-free receiver can receive the message sent by a fault-free sender by using FTVC. Lemma 2 shows that a fault-free receiver can detect that the sender did not send a message to it by using FTVC. Theorem 2 proves the correctness of FTVC. Lemma 3 states that all consistent vertices in an IG-tree are common after the voting function VOTE_{SC} is applied to an IG-tree. By the definition of a common frontier, Lemma 4 shows the existence of a common frontier in an IG-tree. Based on the frontier lemma [8], Theorem 3 shows that the root of a fault-free processor's IG-tree is common. Finally, Theorem 4 proves that the SCMIX is correct under the constraints on failures stated in Section 2.

Lemma 1. *Using FTVC, fault-free receiver R can receive message m sent by fault-free sender S if* $c > 2P_a + P_d$.

Proof. Using FTVC, fault-free sender S sends c copies of m to R through c disjoint paths. According to the path information and transfer rules presented in Section 3.1, each dormant faulty relay can drop at most one message. In the worst case, R receives at least $c - P_d$ messages sent by S. By hypothesis, we know that $c - P_d > 2P_a$. Therefore, R can decide the message sent by S when the majority vote MAJ is applied to these $c - P_d$ messages. □

Lemma 2. *Using FTVC, fault-free receiver R can detect that sender S did not send a message to it if* $c > 2P_a + P_d$.

Proof. When S does not send a message to R, each fault-free immediate successor of S (along the disjoint paths between S and R) will relay the symbol $\varnothing$ to R. In the worst case, R receives at least $c - (P_d - 1)$ messages of value $\varnothing$. By hypothesis, we know that $c - (P_d - 1) > 2P_a$. Hence, the output

of the majority vote MAJ is $\varnothing$, and R notices that S did not send a message to it. □

Theorem 2. *FTVC does remove the influence of a faulty relay if* $c > 2P_a + P_d$.

Proof. By Lemmas 1 and 2, the message received by R is free from the influence of a faulty relay; thus, the theorem is proved. □

Lemma 3. *All consistent vertices are common after* VOTE_{SC} *is applied to an IG-tree if* $n > \max\{mP_a + P_d, 3P_a + P_d\}$.

Proof. Suppose that $k = \max\{m, 3\}$. Each consistent vertex σ of an IG-tree can be proven to be common in the following cases.

Case 1 (σ is a leaf). Fault-free and dormant faulty senders always send identical messages to all receivers. Hence, σ is common after VOTE_{SC} is applied to σ.

Case 2 (σ is at depth $i, 1 \leqslant i \leqslant t$).

Case 2.1. σ has at least $k^*(t - i + 1) + [(n-1)\bmod k]$ children, each of which has a stored value $\mathscr{A}$. By condition $c2$ of VOTE_{SC} stated in Appendix D, the original value stored at σ, namely $\text{val}(\sigma)$, is used as the output of VOTE_{SC}; thus, σ is common.

Case 2.2. σ has $j(< k^*(t - i + 1) + [(n-1)\bmod k])$ children, each of which has a stored value $\mathscr{A}$. According to the structure of the IG-tree, σ has $n - i$ children. By hypothesis, we have $n - P_a - P_d > 2P_a$. Since $t \geqslant P_a$, we have $n - i \geqslant n - t \geqslant n - P_a$; moreover, $j \leqslant P_d$, we can write $n - i - j > 2P_a$. Hence, by condition $c3, c4$ or $c5$ of VOTE_{SC} stated in Appendix D, σ is common. □

Lemma 4. *A common frontier does exist in the IG-tree.*

Proof. By definition, an IG-tree is a tree of depth $t + 1$. Since the maximum number of arbitrary faulty processors is $P_a(\leqslant t)$, each root-to-leaf path has at least one consistent vertex. By Lemma 3, a consistent vertex is common. Therefore, a common frontier does exist in an IG-tree. □

Theorem 3. *The root of a fault-free processor's IG-tree is common.*

Proof. Let $k = \max\{m, 3\}$. According to the structure of the IG-tree, root **E** has n children. If no arbitrary faulty processor exists in the network, namely $P_a = 0$, the message passing is influenced by dormant faulty processors only. These influences are removed by using AR_{SC}; therefore, **E** is common. Generally, suppose that some senders in the network are subjected to arbitrary faults, namely $P_a > 0$. By hypothesis, we have,

$$n - P_a - P_d > (k-1)P_a.$$

Since $P_a > 0$, we can write,

$$n > n - P_a, \Rightarrow\ n - P_d > n - P_a - P_d > (k-1)P_a,$$
$$\Rightarrow\ n > (k-1)P_a + P_d.$$

Since $(k-1) > 2$ $(k = \max\{m, 3\})$, by majority concept, **E** is common after VOTE_{SC} is applied. □

Theorem 4. *SCMIX does solve the SC problem with mixed failure types if* $n > \max\{mP_a + P_d, 3P_a + P_d\}$ *and* $c > 2P_a + P_d$.

Proof. By Theorem 3, the agreement condition is satisfied. That SCMIX satisfies the strong validity condition is shown as follows. Let $V_{FF} \subseteq V$ be the initial values of the fault-free processors. If $V_{FF} = V$, then strong validity is trivially satisfied. On the other hand, if $V_{FF} \neq V$, then there are at most $k-1 (k = \max\{m, 3\})$ values among the fault-free processors. Since $n > kP_a + P_d$, there are at least $n - P_a - P_d > (k-1)P_a$ fault-free processors. Thus, for at least one value in V_{FF}, there are more than P_a fault-free processors with that initial value.

When a fault-free processor applies VOTE_{SC} to the root **E** of its IG-tree, it first applies VOTE_{SC} to the n children of **E**. By Lemma 3, each depth 1 vertex that corresponds to a fault-free processor outputs the original value stored at that vertex, which is the initial value of the corresponding processor. By the above observation, some value in V_{FF} is output by VOTE_{SC} for more than P_a vertices at depth 1 because $n > kP_a + P_d$ and the influence of the dormant faults is removed by AR_{SC}. Since all fault-free processors agree on this value, strong validity is satisfied. Thus, the theorem is proven. □

4.2. Complexity

The SC problem with mixed failure types is solved by SCMIX that is based on the oral message model [3]. In this model, all fault-free processors should exchange *enough* messages in order to reach a common value for the SC problem. Thus, the time for message passing dominates the entire execution of SCMIX and the complexity analysis of SCMIX is focused on *message complexity*. The complexity of SCMIX is defined in terms of: (1) the number of rounds required, (2) the number of messages required, and (3) the number of faulty components allowed. In this section, we prove that SCMIX is optimal. It uses the minimum number of rounds and messages, and tolerates a maximum number of faulty components.

To solve the SC problem with mixed failure types in a generalized network, Theorem 5 shows that SCMIX requires $t+1$ rounds and $(t+1)cn^2$ messages. Theorem 6 shows that SCMIX can solve the problem by using a minimum number of rounds and messages, and Theorem 7 proves that SCMIX can tolerate a maximum number of allowable faulty processors.

Theorem 5. *SCMIX requires* $t+1$ *rounds and* $(t+1)cn^2$ *messages to solve the SC problem with mixed failure types if* $n > \max\{mP_a + P_d, 3P_a + P_d\}$ *and* $c > 2P_a + P_d$.

Proof. The message passing is required in the message exchange phase only; thus, SCMIX requires $t+1$ rounds and this number is the minimum as shown by Fischer and Lynch [16]. In each round, a processor packs the values stored at the last level of the IG-tree to a message, and uses FTVC (c copies of the message are sent to a processor) to broadcast the message to all processors. Hence, there are cn^2 messages generated in each message exchange round. Therefore, the total number of messages required by SCMIX is $(t+1)cn^2$. By Theorem 4, SCMIX can enable all fault-free processors to reach an agreement. Hence, the theorem is proven. □

Theorem 6. *SCMIX solves the SC problem with mixed failure types by using a minimum number of rounds and messages.*

Proof. If the system's fault status is unknown, then $t+1$ rounds are proven to be the lower bound on message passing for reaching an agreement [16]. By Theorem 5, at least $(t+1)cn^2$ messages are required to reach a common value. Hence the theorem is proven. □

Theorem 7. *The total number of allowable faulty processors by SCMIX, namely* $P_a + P_d$, *is maximum if* $n > \max\{mP_a + P_d, 3P_a + P_d\}$ *and* $c > 2P_a + P_d$.

Proof. As stated in Section 1, a protocol for the SC problem with mixed failure types does exist if the constraints on failures, namely $n > \max\{mP_a + P_d, 3P_a + P_d\}$ and $c > 2P_a + P_d$, hold. Otherwise, an agreement cannot be reached. If $P_a + P_d$ is not the maximum number of allowable faulty processors, then other constraints on failures should exist, namely $n \leqslant \max\{mP_a + P_d, 3P_a + P_d\}$ or $c \leqslant 2P_a + P_d$. However, this stands in contradiction with Theorem 1. Thus, the theorem is proven. □

5. Conclusion

SCMIX is a protocol for solving the SC problem with mixed failure types in a network proven in Theorem 4. We have shown the conditions for an agreement, namely the number of processors required and the connectivity required as stated in Theorem 1. Since SCMIX is based on the general assumptions of mixed failure types and generalized network topology, the protocol of [6] is a special case of SCMIX as shown in Table 1. From the previous discussion, we can present the following results.

1. In solving the SC problem for a nonfully connected network, SCMIX is optimal in terms of the number of rounds required, the number of messages required, and the number of faulty components allowable as proven in Theorems 5–7.
2. SCMIX does not require a priori knowledge of processor fault status.
3. SCMIX is designed to solve the SC problem with the most general assumptions on processors as shown in Table 1.
4. The FTVC protocol provides a reliable communication mechanism for removing the influence of faulty relays.

Since SCMIX was originally designed for handling processor faults, it cannot tolerate the maximum number of allowable faulty components when processors and links can both fail [14]. Our future work will focus on improving SCMIX to where it can solve the SC problem with mixed failure types in both processors and links.

Appendix A. Path information

The path information about each sender and receiver pair is distributed to the reply processors between sender and receiver. Each relay processor P maintains tuple (*receiver, sender, predecessor, successor*) path information such that the path $\langle$*predecessor, P, successor*$\rangle$ constitute a subpath of the path from the *sender* to the *receiver*. The sender and receiver also need the c neighbors along a prescribed set of processor-disjoint paths. The sender will send c copies of the message formatted (*receiver, sender, message*) along the c predefined paths to the receiver during each round of message passing.

Appendix B. Transfer rules

The transfer rules obeyed by a relay processor P are defined as follows:

$R1$: According to the path information described above, P only relays messages to its predefined immediate successor if it receives them from its predefined immediate predecessor.

$R2$: Let P be a predefined immediate successor of the sender S. If after time $T_k + T_{sp}$, P has not received a message from S, then P will relay the symbol $\varnothing$ to its predefined immediate successor, where T_k is the starting time of the kth round of the message exchange phase, and T_{sp} is the upper bound on communication time between S and P.

Semantically, $R1$ indicates that a fault-free relay receives messages *only* from its predefined immediate predecessor and sends messages only to its predefined immediate successor. $R2$ is proposed to help R to determine the status of S.

Table 1
The constraints on failures for the SC protocols

Assumption	Arbitrary fault		Dormant fault		Mixed fault	
Result (protocol)	Fully connected network	Nonfully connected network	Fully connected network	Nonfully connected network	Fully connected network	Nonfully connected network
Neiger [6]	$n > \max\{mP_a, 3P_a\}$	NA	$n > \max\{mP_d, 3P_d\}$	NA	$n > \max\{m(P_a + P_d), 3(P_a + P_d)\}$	NA
SCMIX	$n > \max\{mP_a, 3P_a\}$	$n > \max\{mP_a, 3P_a\}$ $c > 2P_a$	$n > P_d$	$n > P_d$ $c > P_d$	$n > \max\{mP_a + P_d, 3P_a + P_d\}$	$n > \max\{mP_a + P_d, 3P_a + P_d\}$, $c > 2P_a + P_d$

NA: not applicable.

Since the network is synchronous, the starting time of each round and the upper bound on each link's communication time can be predefined by each fault-free processor [13]. At T_{sp} after the starting time of the kth round, namely $T_k + T_{sp}$, the predefined immediate successor P of S should have the message sent by S; otherwise, it knows that S is faulty. When P receives no message from S, it relays the symbol $\varnothing$ to its predefined immediate successor to reflect the fault status of S. The properties of path information and transfer rules can be found in current network path components such as ATM-based networks [18].

Appendix C. Conventional majority vote for mixed failure types

Fig. 6 shows that an agreement cannot be reached in the network shown in Fig. 5 when conventional majority voting is used. When conventional majority voting is applied to the internal vertex *AB* shown in Fig. 6, the output of the voting is still contaminated by vertex *ABG* that corresponds to the arbitrary faulty sender G (it sends different messages to different receivers). Although the vertices correspond to dormant faulty senders, *ABD*, *ABE*, and *ABF*, they are not counted when the vote is taken [10,7,11], and the number of children related to the fault-free senders in vertex *AB* is not greater than that of the arbitrary faulty senders. Hence, the voting result is dominated by the value stored in vertex *ABG*. Consequently, the fault-free processors, A, B and C, are unable to reach an agreement when conventional majority voting is used.

Appendix D. VOTE_{SC}

VOTE_{SC} only counts the non-$\mathscr{A}$ values (excluding the last level of the IG-tree). Suppose that $k = \max\{m, 3\}$. For all vertex σ at depth i of an IG-tree, the output of VOTE_{SC} depends on the following conditions:

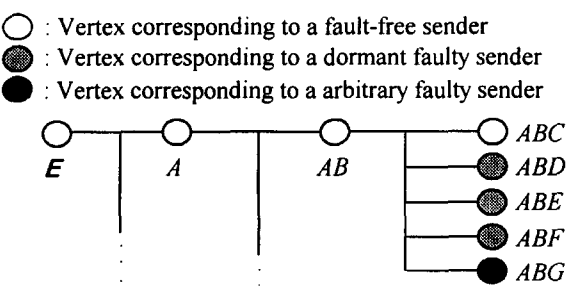


Fig. 6. A subtree of the IG-tree for the network shown in Fig. 5.

VOTE$_{SC}(\sigma)$
begin
if σ is a leaf /* condition $c1$ */
then output val(σ)
else begin
let v be the most common value of VOTE$_{SC}(\sigma p)$, for all child p of vertex σ stored at depth i of IG-tree, and w be the number of copies of value v;
let $x = k * (t - i + 1) + [(n - 1) \bmod k]$;
if $w \geqslant x$ and $v = \mathscr{A}$ /* condition $c2$ */
then output val(σ)
else if $v \neq \mathscr{R}\mathscr{A}_j$, where $1 \leqslant j < t$ /* condition $c3$ */
then output v
else if $v = \mathscr{R}\mathscr{A}_1$ /* condition $c4$ */
then output value $\mathscr{A}$
else if $v = \mathscr{R}\mathscr{A}_j$ and $j \neq 1$ /* condition $c5$ */
then output $\mathscr{R}\mathscr{A}_{j-1}$
end
end.

Note that if there is more than one most common value in conditions $c3, c4$, and $c5$, then the value returned is the one that appears first in any predefined ordering of the values of $V(V = \{v_1, v_2, \ldots, v_m\})$. All fault-free processors use the same ordering. If the most common value is not unique, the value returned is the one that appears first in any fixed enumeration of the values in V. Conditions $c1$ and $c3$ are similar to conventional majority voting. The other three conditions are used to handle cases of mixed failure types. Semantically, conditions $c4$ and $c5$ are used to report the existence of an absentee. When a majority of processors report that an absentee exists, VOTE$_{SC}$ returns the value $\mathscr{A}$ or $\mathscr{R}\mathscr{A}_{j-1}$ to represent the event. As mentioned in Section 3.3, VOTE$_{SC}$ uses val(σ) as the output if condition $c2$ is satisfied.

When VOTE$_{SC}$ is applied to the vertex AB shown in Fig. 5, condition $c2$ of VOTE$_{SC}$ is satisfied and the original value stored in vertex AB is used as the output of VOTE$_{SC}$. Therefore, the influence of the faulty processor G is removed by using VOTE$_{SC}$ and all fault-free processors can reach a common value '1' after the decision making phase.

References

[1] M. Barborak, M. Malek, A. Dahbura, The Consensus Problem in Fault-Tolerant Computing, ACM Computing Surveys 25 (2) (1993) 171–220.

[2] D. Dolev, M.J. Fischer, R. Fowler, N.A. Lynch, H.R. Strong, An efficient algorithm for Byzantine agreement without authentication, Inform. Comput. 52 (1982) 257–274.

[3] L. Lamport, R. Shostak, M. Pease, The Byzantine Generals Problem, ACM Trans. Prog. Lang. Syst. 4 (3) (1982) 382–401.
[4] H.G. Molina, F. Pittelli, S. Davidson, Applications of Byzantine Agreement in Database Systems, ACM Trans. TODS 11 (1) (1986) 27–47.
[5] M. Pease, R. Shostak, L. Lamport, Reaching agreement in presence of faults, J. ACM 27 (2) (1980) 228–234.
[6] G. Neiger, Distributed Consensus revisited, Inform, Process. Lett. 49 (1994) 195–201.
[7] F.J. Meyer, D.K. Pradhan, Consensus with dual failure modes, IEEE Trans. Parallel Distrib. Syst. 2 (2) (1991) 214–222.
[8] A. Bar-Noy, D. Dolev, C. Dwork, R. Strong, Shifting gears: Changing algorithms on the fly to expedite Byzantine agreement, Proceedings of the Symposium on Principle of Distributed Computing 1987, pp. 42–51.
[9] D. Dolev, The Byzantine generals strike again, J. Algorithms 3 (1) (1982) 14–30.
[10] P. Lincoln, J. Rushby, A formally verified algorithm for interactive consistency under a hybrid fault model, Proceedings of the Symposium on Fault-Tolerate Computing Toulouse, 1993, pp. 402–411.
[11] P. Thambidurai, Y.-K. Park, Interactive Consistency with Multiple failure modes, Proc. Symp. on Reliable Distributed Systems Columbus, OH, 1988, pp. 93–100.
[12] K. Shin, P. Ramanathan, Diagnosis of processors with Byzantine faults in a distributed computing systems, Proceedings of the Symposium on Fault-Tolerate Computing, 1987, pp. 55–60.
[13] M. Fischer, M. Paterson, N. Lynch, Impossibility of distributed consensus with one faulty process, J. ACM 32 (4) (1985) 374–382.
[14] K.Q. Yan, Y.H. Chin, S.C. Wang, Optimal agreement protocol in malicious faulty processors and faulty links, IEEE Trans. on Knowledge and Data Engrg. 4 (3) (1992) 266–280.
[15] N. Deo, GRAPH THEORY with Applications to Engineering and Computer Science, Prentice-Hall, Englewood Cliffs, NJ, 1974.
[16] M. Fischer, N. Lynch, A lower bound for the assure interactive consistency, Inform. Process. Lett. 14 (4) (1982) 183–186.
[17] H.S. Siu, Y.H. Chin, W.P. Yang, A note on consensus on dual failure modes, IEEE Trans. Parallel Distrib. Syst. 3 (1996) 230–255.
[18] R. Handel, M.N. Huber, Integrated Broadband Networks: An Introduction to ATM-based Networks, Addison-Wesley, Reading, MA, 1991.