Byzantine Agreement in the Presence of Mixed Faults on Processors and Links

Hin-Sing Siu, Yeh-Hao Chin, *Senior Member*, *IEEE Computer Society*, and Wei-Pang Yang, *Senior Member*, *IEEE Computer Society*

Abstract—In early stage, the Byzantine agreement (BA) problem was studied with single faults on processors in either a fully connected network or a nonfully connected network. Subsequently, the single fault assumption was extended to mixed faults (also referred to as *hybrid fault model*) on processors. For the case of both processor and link failures, the problem has been examined in a fully connected network with a single faulty type, namely an arbitrary fault. To release the limitations of a fully connected network and a single faulty type, the problem is reconsidered in a *general network*. The processors and links in such a network can both be subjected to different types of fault simultaneously. The proposed protocol uses the minimum number of message exchanges and can tolerate the maximum number of allowable faulty components to make each fault-free processor reach an agreement.

Index Terms—Byzantine agreement, fault-tolerant distributed system, hybrid fault model, general network, synchronization.

1 Introduction

Reaching agreement in the presence of faults is one of the most important problems in designing a fault-tolerant distributed system. Achieving such a goal is called the Byzantine Agreement (*BA*) problem [13], [18], and the goal of the *BA* is to make the fault-free processors reach a common agreement, even if certain components (both processors and links) fail. The common agreement of fault-free processors should be free from the influence of the faulty components. With the common agreement, some applications can then be achieved. Some examples are data processing in a distributed database system [17], or the clock synchronization problem [4].

In practice, most of the network topologies may not be fully connected. Processors and links can both be subjected to different types of failures simultaneously, also referred to as the *hybrid fault model* [14], [16], [24]. A fault-free processor does not know which component in the network is faulty. We call such a network model a *general network*. However, none of the existing protocols is designed for solving the *BA* problem in a general network [2], [3], [5], [6], [7], [9], [13], [14], [16], [24], [27]. Table 1 summarizes the assumptions of the relevant *BA* protocols.

As indicated in Table 1, the two broad classes of processor faults, namely *arbitrary* and *dormant*, are termed by Meyer and Pradhan [16]. An arbitrary fault can exhibit arbitrary (including malicious) behavior, while a dormant fault reflects the case when faults consists merely of omission of messages or delay in sending or relaying messages. In the arbitrary-resilient *BA* protocols [9], [13], all faults are treated as arbitrary faults, even though some faults may be subjected to dormant faults, such as crash or fail-stop faults. This treatment ignores that the faulty behaviors of dormant faults is *better* than that of arbitrary faults. When a dormant fault exhibits its faulty behavior, it can be detected and ignored by all fault-free processors. Thus, the arbitrary-resilient *BA* protocols cannot tolerate the maximum number of faults if the dormant faults maybe existed. On the other hand, the protocols proposed by Christian et al. [7] for dormant faults cannot cope with the arbitrary faults. These observations motivate the study of the *BA* problem under a hybrid fault model [14], [16], [24]. The goal of [14], [16], [24] is to maximize the number of allowable faulty processors when hybrid fault model is considered.

In a fully connected network with reliable communication links, Lincoln and Rushby [14] proposed the *OMH* protocol for solving the *BA* problem under a hybrid fault model. This protocol is the corrected version of Thambidurai and Park's *Z* protocol [24]. Both protocols assume that the number of processors subjected to arbitrary faults must be known prior to the execution of the protocols. However, this requirement violates the general assumption of the *BA* problem—that a fault-free processor does not know which component is faulty [14]. Moreover, Shin and Ramanathan [21] found that it is impractical to run diagnostics to detect all arbitrary faults in a network. Furthermore, Meyer and Pradhan [16] also indicated that this kind of protocol is unable to reach an agreement among processors when the number of arbitrary faults is overestimated (or underestimated).

Meyer and Pradhan [16] proposed two protocols, Mixed and Mixed-Sum, for solving the *BA* problem with hybrid fault model on processors. They are designed for any network topology. The Mixed protocol, similar to the *OMH* protocol, still requires that the number of arbitrary faults must be known prior to the execution of the protocol, while the goal of the Mixed-Sum protocol is to remove such a limitation. They stated

• *H.-S. Siu is with the Department of Industrial Engineering and Management, MingChi Institute of Technology, Taipei, Taiwan 24306, Republic of China. E-mail: hssiu@ccsun.mit.edu.tw.*
• *Y.-H. Chin is with the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 30043, Republic of China.*
• *W.-P. Yang is with the Deparment of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan 30050, Republic of China.*

Manuscript received 12 Mar. 1996.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 100152.

TABLE 1
THE DIFFERENT ASSUMPTIONS AMONG THE PREVIOUS WORKS OF THE *BA* PROBLEM

Assumptions	Network Topology			Processor Faults			Link Faults	
Previous Works	Fully Connected	Nonfully Connected	Broadcast*	arbitrary	dormant	hybrid	arbitrary	hybrid
Lamport et al. [13]	√			√				
Dolev [9]		√		√				
Christian et al. [7]	√			√	√			
Thambidurai and Park [24]	√					√		
Meyer and Pradhan [16]		√				√		
Lincoln and Rushby [14]	√					√		
Babaoglu and Drummond [3]			√	√			√	
Yan and Chin [26]	√						√	
Yan et al. [27]	√			√			√	

* *This network model is not discussed in the paper.*

that their protocol can tolerate any fault in a system, provided $n > 3P_a + P_d$ and $c > 2P_a + P_d$, where n is the total number of processors, c is the system connectivity, P_a is the number of arbitrary faults, and P_d is the number of dormant faults. Under a hybrid fault model, these are the optimal constraints on failures for the *BA* problem. However, the bound on the number of allowable faulty processors, namely $n > 3P_a + P_d$, for the MIXED-SUM protocol is overestimated.[1] The main reason of the overestimation is that the *traditional majority vote* is used by MIXED-SUM for handling the hybrid fault model. In order to reach the optimal bound on number of faults, therefore, a *new* voting method should be proposed for solving the *BA* problem under a hybrid fault model.

Most of the earlier work for the *BA* problem is designed for handling the processor failures only [7], [9], [13], [14], [16], [24], as indicated in Table 1. In practice, however, a link can also fail [16], [19], [20], [26], [27]. Martin [15] found that links throughout the world occasionally failed. Therefore, many algorithms have been proposed to diagnose the link failures [1], [20]. For example, in the *Common Channel Signaling* (*CCS*) network, based on Signaling System No. 7 (SS7), protocol has a dedicated algorithm to monitor the error of communication links [20]. Traditionally, such a link fault was treated as a processor fault [24]. The treatment ignores the fact that the processor connected with the faulty link is fault-free (i.e., called *innocent processor* [27]); hence, an innocent processor does not involve an agreement under the treatment. Such a treatment contradicts the definition of the *BA* problem that requires all fault-free processors to reach an agreement. From this standpoint, a link failure should be treated differently from the case of a processor failure.

Similar to the discussion of processor faults, the type of link failures can also be classified with respect to the behavior of the failure into two disjoint sets: *dormant* and *arbitrary fault*. The content of a message may be dropped, but is not contaminated by the dormant faulty link. On the other hand, an arbitrary faulty link may exhibit unrestrained behaviors. Although several protocols have been proposed for solving the *BA* problem with the assumption on link failures (link failures only [26] or both processor and link failures [27]), these protocols consider fully connected networks with arbitrary faults only. Hence, when these protocols are applied to Fig. 1, an agreement still cannot be reached among the fault-free processors.

The *BA* problem is reexamined in a general network. The proposed protocol can solve the *BA* problem with the following assumptions:

1) Each fault-free processor does not require prior information of the system's faulty status.
2) The network topology is not necessarily fully connected.
3) Both processors and links are subjected to mixed faults in the network.

The protocol designed for solving the *BA* problem in a general network is called a *generalized protocol for the BA problem* (GPBA). GPBA uses the minimum number of message exchanges to make each fault-free processor reach a common agreement. The number of tolerable faulty components of GPBA is greater than that of the existing protocols when the general network is considered.

2 THE DEFINITIONS AND CONDITIONS FOR AN AGREEMENT

The *BA* problem is considered in a *synchronous* network. In such a network, the bounds on the processing and communication delays of fault-free components are finite [10], [23], [25]. Therefore, a processor executing the proposed protocol will have received correct messages from all other fault-free processors at a predictable point in time. In contrast, Fischer et al. [12] found that agreement in an *asynchronous* network is impossible even if only one processor has failed, and the failure is a crash failure. Therefore, a synchronous network is assumed. The parameters in a synchronous network are assumed as follows:

- N: The set of all processors, the processor's identifier is unique, and $n = |N|$.
- S: The *source*, and $S \in N$.
- V: The set of all possible values.
- v_S: The initial value of S to be broadcast to all other processors, and $v_S \in V$.

1. We found that the correct constraint on number of processors required is $n > \lfloor (n-1)/3 \rfloor + 2P_a + P_d$ [22].

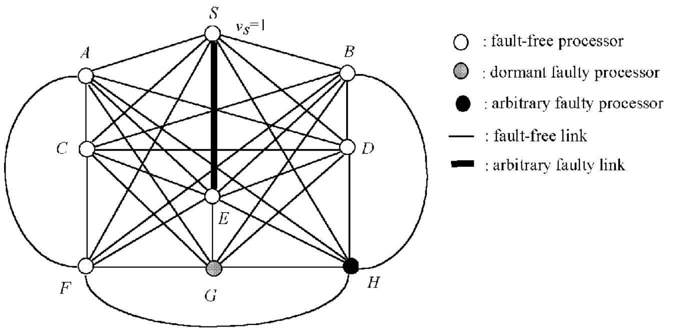
S
v_s=1
A
B
C
D
E
F
G
H
○ : fault-free processor
● : dormant faulty processor
● : arbitrary faulty processor
— : fault-free link
▬ : arbitrary faulty link

Fig. 1. A network with mixed faults on both processors and links (n = 9 and c = 6).

- c: The *connectivity* of the underlying network. Due to the Menger theorem [8], at least c disjoint paths exist between any pairs 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.
- P_a: The number of processors subjected to the arbitrary fault.
- P_d: The number of processors subjected to the dormant fault.
- L_a: The number of links subjected to the arbitrary fault.
- L_d: The number of links subjected to the dormant fault.
- $\varnothing$, A, RA$_i$: These values are used to remove the influence of a faulty processor that does not send its messages; the formal descriptions of these values will be presented in Section 3, and $\varnothing$, A, and RA$_i \notin V$, where $1 \le i < \lfloor (n-1)/3 \rfloor$.

Let processor S be the *source* and v_s be the initial value of S to be broadcast to all other processors. The goal of GPBA is to make each fault-free processor agree on a common value broadcast by the source. After the execution of GPBA, the common value of the fault-free processors shall be the value defined in the following conditions:

1) *Agreement*: All fault-free processors agree on the same common value v, and
2) *Validity*: If the source is fault-free, then the common value v should be the initial value v_s of the source, i.e., $v = v_s$.

In a general network, a processor does not know the faulty status of another component, each processor requires t + 1 rounds to exchange the messages for reaching an agreement [13], where $t = \lfloor (n-1)/3 \rfloor$. Fischer et al. [11] also pointed out that the t + 1 rounds are the minimum number of message exchange rounds required to have an agreement when the network's faulty status is unknown. Therefore, the minimum number of rounds required by GPBA is t + 1.

Essentially, the fault tolerance capabilities of a network depend on the total number of processors and the network topology (connectivity) [9]. For example, every faulty component can prevent the fault-free processors from achieving an agreement if the network topology is a bus. To generalize, the complete characterizations of constraints on failures for every network are discussed below. The Byzantine agreement can be achieved in a network if:

1) $n > 3P_a + P_d$, and
2) $c > 2P_a + P_d + 2(L_a + L_d)$.

The first constraint specifies the number of processors required. After the influences of the dormant faulty processors are removed, an agreement can be reached if $n - P_d > 3P_a$, namely $n > 3P_a + P_d$, as stated in [16]. On the other hand, the second constraint specifies the required connectivity. In each message exchange round, every processor sends its message to other processors. In order to decide whether a processor has sent out its message, the total number of link failures must be less than half of $c - 2P_a - P_d$, namely $c > 2P_a + P_d + 2(L_a + L_d)$. Otherwise, such a goal cannot be reached. Thus, the conditions for achieving an agreement in a general network can be proven (due to the limited space, the proofs of the lemmas and theorems in the paper are omitted).

THEOREM 1. *For any network, a protocol for the BA problem does exist if* $n > 3P_a + P_d$ *and* $c > 2P_a + P_d + 2(L_a + L_d)$.

From the above observation, the types of link failures can be treated as a single failure type. For ease of discussion, we still use the classification of link failures. By Theorem 1, the maximum number allowable of faulty components by GPBA is $P_a + P_d + L_a + L_d$ if $n > 3P_a + P_d$ and $c > 2P_a + P_d + 2(L_a + L_d)$, as stated in Theorem 1. When the number of processors and the connectivity is two or less, there is no problem reaching an agreement [13]; hence, $n > 2$ and $c > 2$ are assumed.

3 Basic Concept and Approaches

In a general network, a processor can be played as a *sender*, *receiver*, or *relay*, depending on the flow of a message. A message sent from a sender to a receiver may be passed through some *intermedium* (relay processors and links). The message may be contaminated by either the sender (dormant or arbitrary fault), some intermedium, or both. In order to solve the *BA* problem in such a network model, GPBA therefore should completely remove the influence caused by these faulty components. In order to reduce the number of message exchange rounds required, as stated in [14], [16], [24], [27], GPBA removes the influence caused by a faulty intermedium first, then removes the influence caused by a dormant faulty sender and, finally, removes the influence caused by an arbitrary faulty sender.

GPBA is based on the *oral message model* [13]. There are two phases in GPBA: *the message exchange phase* and *the decision making phase*. The message exchange phase consists of $\lfloor (n-1)/3 \rfloor + 1$ rounds to collect the messages and the decision making phase computes the common value for an agreement, as shown in Fig. 2. GPBA combines the following approaches to solve the *BA* problem in a general network:

- **The *fault-tolerant virtual channel* (FTVC) protocol**: At the start of the kth round of message exchange, say r_k^-, each sender uses FTVC to broadcast its messages to all receivers, as shown in Fig. 2. The goal of FTVC is to remove the influence caused by a faulty intermedium.

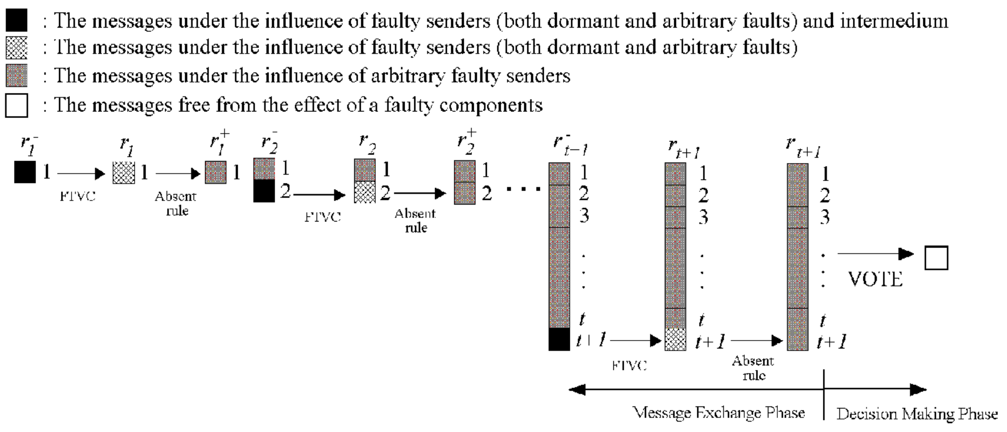
: The messages under the influence of faulty senders (both dormant and arbitrary faults) and intermedium
: The messages under the influence of faulty senders (both dormant and arbitrary faults)
: The messages under the influence of arbitrary faulty senders
: The messages free from the effect of a faulty components
FTVC
Absent rule
VOTE
Message Exchange Phase
Decision Making Phase

Fig. 2. The basic approaches of GPBA.

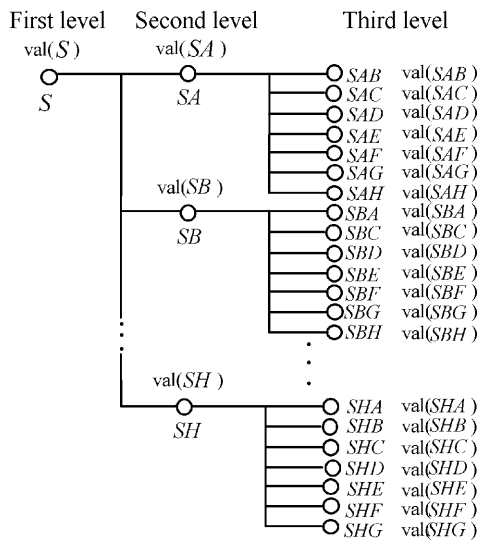

Fig. 3. An example of an IG-tree for the network model shown in Fig. 1.

That means FTVC provides a reliable communication mechanism in a general network so that the message passing in such a network is similar to that in a fully connected network with fault-free communication links. Each receiver collects all of the senders' messages that are free from the influences of a faulty intermedium when FTVC is applied, say r_k, as shown in Fig. 2.

- **The absent rule**: The absent rule is applied at the end of the kth round, namely r_k^+, to remove the influence of the dormant faulty sender, as shown in Fig. 2. Obviously, if a sender has a dormant fault, all fault-free receivers can detect such a fault (no message was received from it) during the entire message exchange phase or at some message exchange rounds. Once a faulty sender is detected, a fault-free receiver will ignore the messages received from it in every subsequent message exchange round.
- **The voting function VOTE**: When the number of rounds reaches $\lfloor (n-1)/3 \rfloor + 1$, the number of received messages collected is enough to remove the influence of an arbitrary faulty sender and to make each fault-free processor able to reach an agreement through a voting scheme VOTE taken in the decision making phase, as shown in Fig. 2.

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) [5], of level $t + 1$, for collecting the received messages. Fig. 3 illustrates the structure of an IG-tree for the network shown in Fig. 2. After the first message exchange round, each fault-free receiver stores the message received from the source, denoted as val(S), at the root S of its IG-tree. In the second round, each sender broadcasts the root's value of its IG-tree to all receivers. If sender B sends a message val(S) to receiver A, A will store the message received from B, denoted as val(SB), at vertex SB of its IG-tree. The vertex SB is said to *correspond* to the sender B. Note that each level of an IG-tree contains a round of received messages and each vertex is labeled by a nonrepeating sequence of processor identifiers. Because the label of an IG-tree is nonrepeating, the root (labeled by the source) has $n - 1$ children and a vertex at the tth level has $n - t$ leaves, as shown in Fig. 3. No repeating processor identifier can avoid the recursive influences made by a faulty processor.

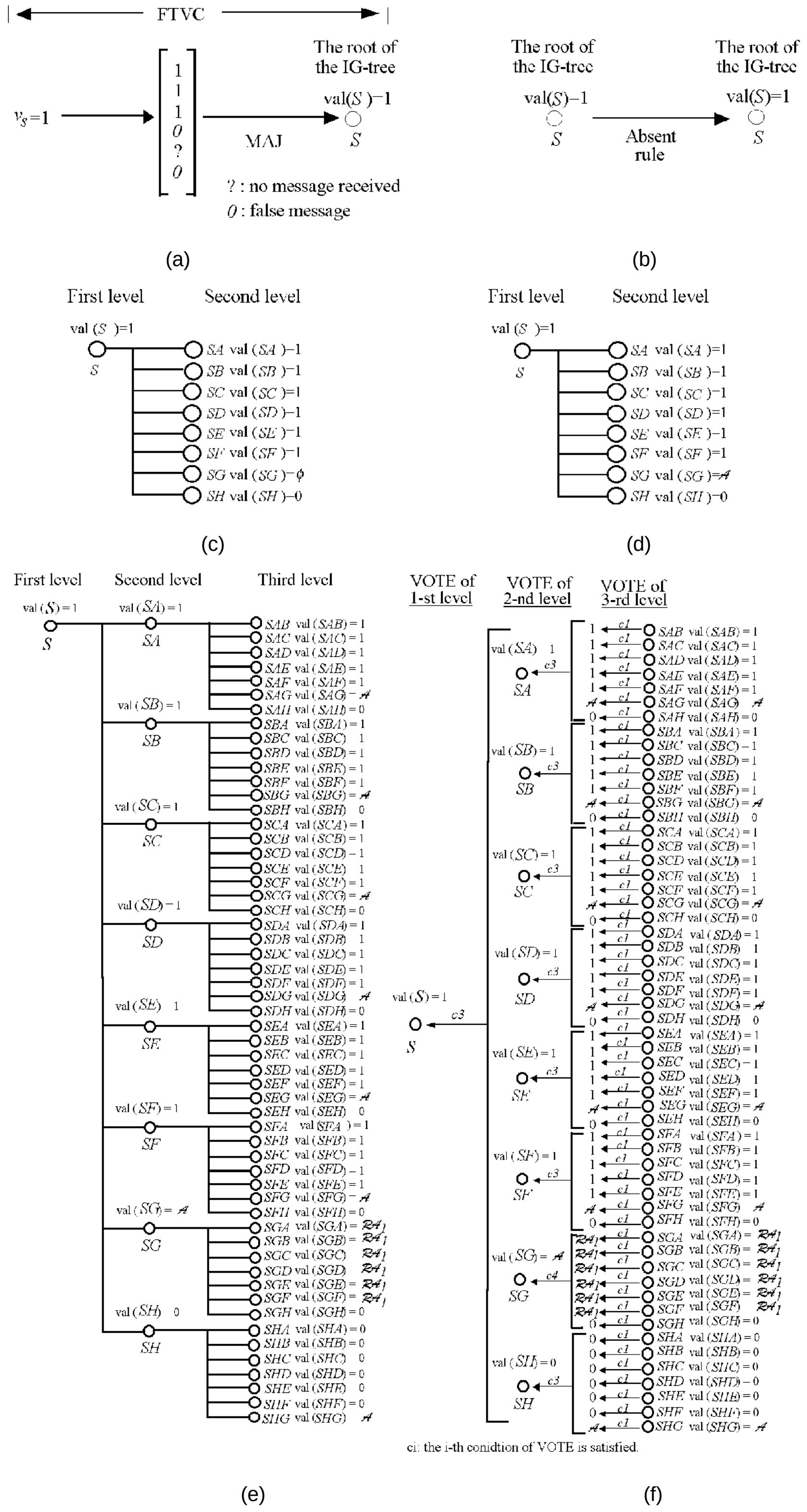

Fig. 4. GPBA solves the *BA* problem in the network model shown in Fig. 1: (a) The source uses FTVC to broadcast its value, (b) applying the absent rule, (c) the second round IG-tree, (d) after the absent rule is applied, (e) after the message exchange phase, (f) applying the VOTE.

Fig. 4 shows an example of GPBA used by a fault-free processor, say *A*, in the network shown in Fig. 1 (the execution in the other fault-free processors is similar). Suppose that the dormant faulty processor *G* does not send any messages during the entire execution of the protocol. In the first round of message exchange, the source *S* uses FTVC to send *c* (= 6) copies of its initial value "1" to each receiver. Each fault-free receiver can receive the message "1" sent by *S* after the majority function MAJ is applied to the *c* received messages, and store the message into the root of its

IG-tree, as shown in Fig. 4a. To remove the influence of a dormant faulty sender, each fault-free receiver then applies the absent rule to the root of its IG-tree as shown in Fig. 4b. Since the source is not in dormant fault, the value stored at the root in each fault-free receiver is unchanged after the absent rule is applied. Figs. 4c and 4d show the second round of message exchange. Using FTVC to exchange the messages, each fault-free receiver stores the messages received from all senders into the second level of its IG-tree, as shown in Fig. 4c. Note that the value stored at vertex *SG* in the IG-tree of each fault-free receiver is ∅. That means a fault-free receiver does not receive any messages from *G* and uses the value ∅ to represent the message of *G*. After the absent rule is applied, each fault-free receiver will use the value A to replace the messages received from *G* as shown in Fig. 4d. Value A will then be relayed to all processors as value RA_1 and value RA_j will be relayed to all processors as value RA_{j+1}, where $1 \le j < t$ (the meanings of value A and RA_j are discussed in Section 3.2). When the number of rounds reaches *three* ($= \lfloor (n-1)/3 \rfloor + 1 = \lfloor (9-1)/3 \rfloor + 1$), the messages are collected by each fault-free processor as shown in Fig. 4e. Finally, each fault-free processor applies the function VOTE onto its IG-tree for computing the common value for an agreement as shown in Fig. 4f. Note that the value A (excluding the last round) is not counted during the time the VOTE is taken. The common value "1" can be reached among the fault-free processors after VOTE is applied.

The detailed descriptions of the above steps for removing the influence caused by the multiple faulty components are presented as follows.

3.1 Step 1: Removing the Influence of a Faulty Intermedium

The function of FTVC is to remove the influence caused by the faulty intermedium between any pairs of processors. We first consider the case of a single sender *S* and a single receiver *R*. *S* uses FTVC to send its message *m* to *R*. Solving this case successfully will enable us to solve the general case in which every sender sends its message to every receiver.

Due to the Menger theorem [8], 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* can be predefined, as stated in [9], [16], and the *paths information* is distributed onto the relay processors between *S* and *R*. The detailed description of the paths information is presented in Appendix A. According to the paths information, a relay processor receives the message (*R*, *S*, *m*) from the predefined immediate predecessor and sends the message to the predefined immediate successor. Since the network is synchronous, the predefined immediate successor *P* of *S* should have the message sent by S after the predefined time interval [12]; otherwise, it knows that either *S*, the link L_{SP}, or both are faulty. When *P* receives no message from *S*, it will relay the symbol ∅ ($\varnothing \notin V$) to its immediate successor along the predefined disjoint path between *S* and *R* to reflect the faulty status. These are the concepts of the *transferring rules* obeyed by each relay processor. The goal of the transferring rules is to make the receiver obtain the correct message sent, directly or indirectly, by the sender. The formal definition of the transferring rules is presented in Appendix B. The properties of path information and transfer rules can be found in current network path components such as ATM-based networks.

By the definition of the transferring rules, normally, *R* receives only one *S's* message from a path between *S* and *R*. If more than one message is received from a path in a message exchange round, then all messages received from the path are discarded. This method can handle the case of the faulty components to fake the messages of predecesors. Therefore, *R* receives at most *c* messages sent by *S*. An arbitrary faulty intermedium between *S* and *R* can influence, at most, one message of these *c* messages sent by *S*, and a dormant faulty intermedium between *S* and *R* can drop, at most, one message of these *c* messages sent by *S*. In the worst case, *R* will receive $c - P_d - L_d$ copies of messages sent by *S*. By applying the majority vote MAJ to these messages, *R* can determine the message sent by *S* if the constraint on connectivity, namely $c > 2P_a + P_d + 2(L_a + L_d)$, holds. MAJ has three possible outcomes:

- Case 1: *m*, if *S* is fault-free.
- Case 2: ∅, if *S* does not send the message to *R*.
- Case 3: *Arbitrary value*, if *S* has an arbitrary fault.

In Case 1, *R* can receive the message *m* 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 ∅ as the message sent by *S* because the *major* number of $c - P_d - L_d$ copies of messages is ∅. The third outcome of MAJ implies that the received message is not only contaminated by the faulty intermedium, but also is contaminated by the arbitrary faulty sender. FTVC is unable to remove the influence of such a case; hence such an outcome of MAJ shall be an arbitrary value.

In [9], [16], the alternative approaches are proposed to remove the influence caused by a faulty intermedium in a nonfully connected network. The approach of [9] assumes that a faulty processor is only subjected to arbitrary faults and a link is fault-free. This approach is inappropriate for hybrid fault model on both processors and links. In [16], an approach for handling hybrid fault model only on processors is proposed. When a receiver cannot determine whether the sender did send a message (regular message) to it, it will send the *inquiry messages* to the sender and will wait for the *denial messages* from the sender to verify the status of the sender. Hence, the length of time occupied by each round of this approach is three times than that of FTVC, which only requires the time for regular message passing.

3.2 Step 2: Removing the Influence of a Dormant Faulty Sender

The second step of GPBA is to remove the influence of a dormant faulty sender. Each fault-free sender should 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 ∅).

In the first round, the source should broadcast its initial value v_s to all receivers; therefore, a fault-free receiver R can always detect that the source is faulty if no message is received from it. In order to satisfy the *Agreement condition* of the *BA* problem [13]—agreement should be reached by every fault-free processor even if no message was sent from the source—R will select the default value, say 0, to replace the source's message if no message is received from the source.

In each subsequent round, these $n - 1$ receivers (except the source) should exchange the message received from the source to compute a common value. A fault-free receiver R can detect that a sender S is faulty if no message was 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 the subsequent rounds will be replaced by value A, and this value will be relayed to all processors as value RA_1. In each subsequent round, the value RA_j will be relayed to all processors as value RA_{j+1} ($A \notin V$ and $\forall i$, $RA_i \notin V$).

Semantically, the value A is represented as an *absentee vote*, while sender S is treated as an *absentee*. Hence, the voting ticket of S is ignored during the decision-making phase. Value RA_i will be interpreted as *the ith time an absent vote reported*. R will report to all processors that S is an absentee; thus, S has no influence on the others when the voting function VOTE is taken in decision-making phase. The approach can be formalized as follows:

Absent rule:

(AR1) When R receives no message directly from S in the rth round, then

1) if $r = 1$ (the first round), P will select the *default value* to replace the incoming message from S (the source); or
2) if $r \neq 1$, all messages received from S at the rth round and any subsequent rounds will be replaced by value A, and this value will be relayed to all processors (except the transmitter) as value RA_1.

(AR2) When R receives the value RA_j, it will relay the value RA_{j+1} to all processors (if any).

3.3 Step 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 the faulty intermedium and the dormant faulty sender. However, the messages are still contaminated by the arbitrary faulty senders. In order to reach an agreement, such an influence shall also be removed in the decision making phase. As mentioned in Section 1, the traditional majority vote is inappropriate in a general network. As a result, we propose a new voting scheme VOTE to remove the influence of the arbitrary faulty senders.

By the constraint on the number of processors required, namely $n > 3P_a + P_d$, if the network eliminates one arbitrary faulty sender, then it can tolerate three more dormant faulty senders because $3(P_a - 1) + (P_d + 3) = 3P_a + P_d$, where $P_a \geq 1$. This phenomenon can be used by VOTE to remove the influence of an arbitrary faulty sender. The basic concept of VOTE is as follows: Let P be a fault-free processor and σ be a vertex at the ith level of P's IG-tree, $1 \leq i \leq t$. If P detects that $3*(t-i+1) + [(n-1) \bmod 3]$ children of σ have value A, it will use the original value stored at σ, namely val(σ), as the output of VOTE for removing the influence of the arbitrary faulty sender as in the above discussion; otherwise, it will use the majority value of children of σ as the output of VOTE.

VOTE 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, if vertex σ corresponds to an arbitrary faulty sender Q, the output of VOTE *may* be contaminated by Q after our approach is applied (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 at the upper level voting if $n > 3P_a + P_d$. Therefore, the fault-free processors can reach an agreement through the concept of *democratic voting*, as shown in Theorem 3. Appendix C presents the formal definition of VOTE.

4 Analysis and Evaluation of GPBA

GPBA uses the approaches stated in Section 3 to remove the multiply 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 the FTVC protocol.
- FTVC_RECEIVE(m, Q): Receive the message m from processor Q by using the FTVC protocol.
- ABSENT_RULE(r): Apply the absent rule to the rth level of the IG-tree.
- VOTE (s): Apply the function VOTE to vertex s.

For completeness, some additional primitives should be presented as follows:

- CREATE(σQ, v): Create the vertex σQ, and set val(σQ) $= v$.
- FOLD(r, m): Fold the rth level of the IG-tree to the message m.
- UNFOLD(m, r): According the structure of the rth level of the IG-tree, unfold the message.
- OUTPUT(v): Output the value v.

Using the above primitives, the formal procedure of GPBA can be stated as follows:

Protocol GPBA (for each processor P)

```
begin
/* Message Exchange Phase */
   /* The first round */
     if $P = S$   /* the source $S$*/
        for $Q \in N - \{S\}$ do
             FTVC_SEND($v_s$, $Q$);
   else
     begin
            FTVC_RECEIVE($v_s$, $S$);
            CREATE($S$, $v_s$);
            ABSENT_RULE(1)
     end;
   /* round 2 to round $t + 1$ */
   for $r = 2$ to $t + 1$ do
   begin
      FOLD($r - 1$, $m$);
```

```
        for $Q \in N - \{S\}$ do
            FTVC_SEND($m$, $Q$);
        for $Q \in N - \{S\}$ do
        begin
            FTVC_RECEIVE($m$, $Q$);
            UNFOLD($m$, $r$);
            for $\sigma \in m$ do
            begin
              $v$ = val($\sigma$);
              CREATE($\sigma Q$, $v$)
            end
        end;
        ABSENT_RULE($r$)
    end;
/* Decision Making Phase */
    VOTE($S$);
    Output(val($S$))
end.
```

4.1 Correctness

The goal of GPBA is to make each fault-free processor reach a common value for an agreement; hence, the correctness of GPBA can be proven from the fact that the common value of each fault-free processor satisfies the conditions of *Agreement* and *Validity*. To reach an agreement in a general network, each fault-free processor should be insulated from the influence of all faulty components. The basic concept of proving the correctness of GPBA is as follows: By using FTVC, GPBA first removes the influence of a faulty intermedium. Then, it makes each fault-free processor reach an agreement after the influence of the faulty sender is removed (both arbitrary and dormant faults).

To prove the correctness of FTVC, the output of MAJ should be proven to be free from the influence of a faulty intermedium. Hence, we shall prove that a fault-free receiver can receive the message sent by a fault-free sender or can detect that the sender did not send a message to it.

LEMMA 1. *Using FTVC, a fault-free receiver R can receive the message m sent by a fault-free sender S if* $c > 2P_a + P_d + 2(L_a + L_d)$.

LEMMA 2. *Using FTVC, a fault-free receiver R can detect that the sender S does not send a message to it if* $c > 2P_a + P_d + 2(L_a + L_d)$.

THEOREM 2. *FTVC does remove the influence of a faulty intermedium in a general network if* $c > 2P_a + P_d + 2(L_a + L_d)$.

Then, we prove that GPBA makes each fault-free processor reach an agreement. Since GPBA is based on the oral message model, some concepts and terminologies used by [5] are used here. A vertex σ is called *common* [5] if the value stored at σ of each fault-free processor's IG-tree is identical. In other words, a common value for an agreement can be reached if the root of each fault-free processor's IG-tree is common. If every root-to-leaf of an IG-tree contains a common vertex, then the collection of the common vertices forms a *common frontier* [5]. To prove an agreement can be reached by GPBA, we define the *consistent vertex* as follows.

Vertex α (= σi) at a fault-free receiver's IG-tree is a consistent vertex if sender i is fault-free or in dormant fault. By the behavior of i, all fault-free receivers receive the identical message sent by i. Although a receiver does not know which vertex is consistent, the consistent vertices do exist since some senders in the network are fault-free or dormant faulty.

The following lemma proves that all consistent vertices of an IG-tree are common:

LEMMA 3. *All consistent vertices are common after VOTE is applied to an IG-tree if* $n > 3P_a + P_d$.

LEMMA 4: *The common frontier does exist in the IG-tree.*

By the frontier lemma of [5], the root of the fault-free processor's IG-tree is common if the common frontier exists on each fault-free processor's IG-tree. The following theorems prove that an agreement can be reached among each fault-free processor.

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

THEOREM 4. *GPBA does solve the BA problem with mixed faults if* $n > 3P_a + P_d$ *and* $c > 2P_a + P_d + 2(L_a + L_d)$.

4.2 Complexity

The complexity of GPBA 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 subsection, we prove that GPBA is optimal. It uses the minimum number of rounds and messages to tolerate the maximum number of faulty components.

THEOREM 5. *GPBA requires* $t + 1$ *rounds and* $O(cn + tcn^2)$ *messages for solving the BA problem in a general network if* $n > 3P_a + P_d$ *and* $c > 2P_a + P_d + 2(L_a + L_d)$, *where* $t = \lfloor (n-1)/3 \rfloor$.

THEOREM 6. *GPBA solves the* BA *problem in a general network by using the minimum number of rounds and messages.*

THEOREM 7. *The total number of allowable faulty components by GPBA, namely* $P_a + P_d + L_a + L_d$, *is maximum if* $n > 3P_a + P_d$ *and* $c > 2P_a + P_d + 2(L_a + L_d)$.

5 CONCLUSION

GPBA is a protocol for solving the *BA* problem in a general network. We have shown the conditions for an agreement in a general network, namely the number of processors required and the connectivity required. Since GPBA is based on general assumptions, the protocols designed for handling the processor failures only or link failures only are the special cases of GPBA, as summarized in Tables 2 and 3. As for the assumptions on both processor and link failures, Yan et al.'s protocol [27] is also a special case of GPBA because their protocol considers arbitrary faults only, as indicated in Table 4.

When only link failures are considered, the constraint on failures by GPBA is $c > 2L_a + L_d$, as shown in Table 3. The number of allowable faulty links of this bound is greater than that of the original bound on GPBA, namely $c > 2(L_a + L_d)$.

TABLE 2
THE CONSTRAINTS ON FAILURES FOR VARIOUS PROTOCOLS ON PROCESSOR FAULTS

Assumption	arbitrary fault		dormant fault		hybrid fault	
Result / Protocol	fully connected network	nonfully connected network	fully connected network	nonfully connected network	fully connected network	nonfully connected network
Lamport et al. [14]	$n > 3P_a$	N.A.	$n > 3P_d$	N.A.	$n > 3(P_a + P_d)$	N.A.
Dolev [9]	$n > 3P_a$	$n > 3P_a$ $c > 2P_a$	$n > 3P_d$	$n > 3P_d$ $c > 2P_d$	$n > 3(P_a + P_d)$	$n > 3(P_a + P_d)$ $c > 2(P_a + P_d)$
Christian et al. [7]	$n > 3P_a$	N.A.	$n > P_d$	N.A.	$n > 3(P_a + P_d)$	N.A.
Thambidurai and Park [24]	$n > 3P_a$	N.A.	$n > P_d$	N.A.	$n > 3P_a + P_d$*	N.A.
Lincoln and Rushby [15]	$n > 3P_a$	N.A.	$n > P_d$	N.A.	$n > 3P_a + P_d$*	N.A.
Meyer and Pradhan [17]	$n > 3P_a$	$n > 3P_a$ $c > 2P_a$	$n > P_d$	$n > P_d$ $c > P_d$	$n > \lfloor (n-1)/3 \rfloor + 2P_a + P_d$	$n > \lfloor (n-1)/3 \rfloor + 2P_a + P_d$ $c > 2P_a + P_d$
GPBA	$n > 3P_a$	$n > 3P_a$ $c > 2P_a$	$n > P_d$	$n > P_d$ $c > P_d$	$n > 3P_a + P_d$	$n > 3P_a + P_d$ $c > 2P_a + P_d$

N.A.: Not Applicable
** The number of arbitrary faulty processors must be known.*

TABLE 3
THE CONSTRAINTS ON FAILURES FOR VARIOUS PROTOCOLS ON LINK FAILURES

Assumptions	arbitrary fault		hybrid fault	
Results / Protocol	fully connected network	nonfully connected network	fully connected network	nonfully connected network
Yan and Chin [26]	$c > 2L_a$	N.A.	$c > 2(L_a + L_d)$	N.A.
GPBA	$c > 2L_a$	$c > 2L_a$	$c > 2L_a + L_d$	$c > 2L_a + L_d$

TABLE 4
THE CONSTRAINTS ON FAILURES FOR VARIOUS PROTOCOLS ON BOTH PROCESSOR AND LINK FAILURES

Assumptions	arbitrary fault		hybrid fault	
Results / Protocol	fully connected network	nonfully connected network	fully connected network	nonfully connected network
Yan et al. [27]	$n > 3P_a$ $c > 2P_a + L_a$	N.A.	$n > 3(P_a + P_d)$ $c > 2(P_a + P_d + L_a + L_d)$	N.A.
GPBA	$n > 3P_a$ $c > 2P_a + L_a$	$n > 3P_a$ $c > 2P_a + L_a$	$n > 3P_a + P_d$ $c > 2P_a + P_d + 2(L_a + L_d)$	$n > 3P_a + P_d$ $c > 2P_a + P_d + 2(L_a + L_d)$

According to FTVC, in the worst case, a fault-free processor has $c - L_d$ copies of messages sent by a sender. Therefore, the influence of faulty links can be removed by using FTVC if $c - L_d > 2L_a$, namely $c > 2L_a + L_d$. Since the processors are fault-free, by the condition *c3* of VOTE, an agreement can be reached among the processors.

From the previous discussion, we can present the following results:

1) Solving the *BA* problem in a general network, GPBA 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 6 and 7.
2) GPBA does not require the faulty status of the system prior to the execution of the protocol.
3) GPBA is designed for solving the *BA* problem with the most general assumptions; all others failure types can be treated as the special cases as shown in Tables 2, 3, and 4.
4) The FTVC protocol can remove the influence of a faulty intermedium. The protocol can be used in other problems of distributed systems, such as clock synchronization and replicated file system [3], to provide a reliable communication mechanism.

By using the absent rule, a fault-free receiver can detect the faulty status of a sender. Thus, GPBA has the capability of *fault detection*. Basically, the fault detection can handle the dormant faults only and the result of fault detection is *local* [3] (the fault detection result is *inconsistent* among all fault-free processors). Our future work consists of combining GPBA with the fault detection for all types of failures in a general network.

APPENDIX

A Paths Information

The paths information of each sender and receiver pair is distributed onto the replay processors between sender and receiver. Each relay processor *P* maintains a tuple (*receiver, sender*, *predecessor*, *successor*) path information such that the path <*predecessor, P, successor*> is 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*) through the *c* predefined paths to the receiver at each time of message passing.

B The Transferring Rules

The transferring rules obeyed by a relay processor *P* are defined in the following:

R1: According to the paths information described above, *P* only relays a message to its predefined immediate successor if *P* receives it from its predefined immediate predecessor.

R2: Let *P* be a predefined immediate successor of the sender *S*. After the time $T_k + T_{SP}$, if *P* does not receive a message from *S*, then *P* will relay the symbol ∅ to its predefined immediate successor, where T_k is the starting time of the *k*th 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 processor *only* receives a message from the predefined immediate predecessor and *only* sends a message to the predefined immediate successor. *R2* is proposed to help *R* to determine the status of *S*. At the time T_{SP} after the starting time of the *k*th round, namely $T_k + T_{SP}$, the predefined immediate successor *P* of *S* should have the message sent by *S*; otherwise, it knows that either *S*, the link L_{SP}, or both have failed. When *P* receives no message from *S*, it will relay the symbol ∅ to its predefined immediate successor to reflect the faulty status.

C VOTE

VOTE only counts the non-*A* values (excluding the last level of the IG-tree). For all vertex σ at the *i*th level of an IG-tree, the output of VOTE depends on the following conditions:

c1: val(σ), if σ is a leaf; or

c2: val(σ), if $1 \le i \le t$ and σ has at least 3 * $(t - i + 1)$ + [$(n - 1)$ mod 3] children, each of which has value *A*; or

c3: *v*, if $1 \le i \le t$, σ has no more than 3 * $(t - i + 1)$ + [$(n - 1)$ mod 3] children, each of which has value *A*, *v* is the most common value of VOTE applied to children of σ, and $v \ne RA_i$; or

c4: *A*, if $1 \le i \le t$, σ has no more than 3 * $(t - i + 1)$ + [$(n - 1)$ mod 3] children, each of which has value *A*, and RA_1 is the most common value of VOTE applied to children of σ; or

c5: RA_{j-1}, if $1 \le i \le t$, σ has no more than 3 * $(t - i + 1)$ + [$(n - 1)$ mod 3] children, each of which has value *A*, and RA_j is the most common value of VOTE applied to children of σ, where $j \ne 1$; or

c6: the default value, if no majority value of children of σ exists.

Conditions *c1*, *c3*, *c5*, and *c6* are similar to the traditional majority vote. The other two conditions are used to handle the case of mixed faults. Semantically, condition *c4* is used to report the existence of an absentee. When the major number of processors reports that an absentee exists, VOTE returns the value *A*, an absentee's vote, to represent such an event. As mentioned in Section 3.3, VOTE uses val(σ) as the output if the condition of *c2* is satisfied.

REFERENCES

[1] J.C. Adams and K.V.S. Ramarao, "Distributed Diagnosis of Byzantine Processors and Links," *Proc. Symp. Distributed Computing Systems*, pp. 562-569, 1989.

[2] O. Babaoglu, "On the Reliability of Consensus-Based Fault-Tolerant Distributed Computing Systems," *ACM Trans. Computer Systems,* vol. 5, no. 2, pp. 394-416, Nov. 1987.

[3] O. Babaoglu and R. Drummond, "Street of Byzantium: Network Architectures for Fast Reliable Broadcasts," *IEEE Trans. Software Eng.*, vol. 11, no. 6, pp. 546-554, June 1985.

[4] M. Barborak, M. Malek, and A. Dahbura, "The Consensus Problem in Fault-Tolerant Computing," *ACM Computing Surveys*, vol. 25, no. 2, pp. 171-220, June 1993.

[5] A. Bar-Noy, D. Dolev, C. Dwork, and R. Strong, "Shifting Gears: Changing Algorithms on the Fly to Expedite Byzantine Agreement," *Proc. Symp. Principle of Distributed Computing*, pp. 42-51, 1987.

[6] P. Berman and J.A. Garay, "Cloture Votes: n/4-Resilient Distributed Consensus in t + 1 Rounds," *Math. Systems Theory*, vol. 26, no. 1, pp. 3-19, 1993.

[7] F. Christian, H. Aghili, and H.R. Strong, "Atomic Broadcase: From Simple Message Diffusion to Byzantine Agreement," *Proc. Symp. Fault-Tolerant Computing*, pp. 200-205, Ann Arbor, Mich., 1985.

[8] N. Deo, *GRAPH THEORY with Applications to Engineering and Computer Science*. Englewood Cliffs, N.J.: Prentice Hall, 1974.

[9] D. Dolev, "The Byzantine Generals Strike Again," *J. Algorithms*, vol. 3, no. 1, pp. 14-30, 1982.

[10] D. Dolev, N. Lynch, S. Pinter, E. Stark, and W. Weihl, "Reaching Approximate Agreement in the Presence of Faults," *J. ACM*, vol. 33, no. 3, pp. 499-516, July 1986.

[11] M. Fischer and N. Lynch, "A Lower Bound for the Assure Interactive Consistency," *Information Processing Let*ters, vol. 14, no. 4, pp. 183-186, June 1982.

[12] M. Fischer, M. Paterson, and N. Lynch, "Impossibility of Distributed Consensus with One Faulty Process," *J. ACM*, vol. 32, no. 4, pp. 374-382, Apr. 1985.

[13] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Trans. Programming Language Systems,* vol. 4, no. 3, pp. 382-401, July 1982.

[14] P. Lincoln and J. Rushby, "A Formally Verified Algorithm for Interactive Consistency Under a Hybrid Fault Model," *Proc. Symp. Fault-Tolerant Computing*, pp. 402-411, Toulouse, France, 1993.

[15] J. Martin, *Telecommunications and the Computer*, third ed. Englewood Cliffs, N.J.: Prentice Hall, 1990.

[16] F.J. Meyer and D.K. Pradhan, "Consensus with Dual Failure Modes," *IEEE Trans. Parallel and Distributed Systems,* vol. 2, no. 2, pp. 214-222, Apr. 1991.

[17] H.G. Molina, F. Pittelli and S. Davidson, "Applications of Byzantine Agreement in Database Systems," *ACM Trans. Database Systems*, vol. 11, no. 1, pp. 27-47, Mar. 1986.

[18] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in Presence of Faults," *J. ACM,* vol. 27, no. 2, pp. 228-234, Apr. 1980.

[19] A. Pelc, "Reliable Communication in Networks with Byzantine Link Failures," *NETWORKS,* vol. 22, no. 5, pp. 441-459, Aug. 1992.

[20] V. Ramaswami, and J.L. Wang, "Analysis of the Link Error Monitoring Protocols in the Common Channel Signaling Network," *IEEE/ACM Trans. Networking*, vol. 1, no. 1, pp. 31-47, Feb. 1993.

[21] K. Shin and P. Ramanathan, "Diagnosis of Processors with Byzantine Faults in Distributed Computing Systems," *Proc. Symp. Fault-Tolerant Computing*, pp. 55-60, 1987.

[22] H.S. Siu, Y.H. Chin, and W.P. Yang, "A Note on Consensus on Dual Failure Modes," *IEEE Trans. Parallel and Distributed Systems,* vol. 7, no. 3, pp. 225-230, Mar. 1996.
[23] N. Suri, M.M. Hugue, and C.J. Walter, "Synchronization Issues in Real-Time Systems," *Proc. IEEE*, vol. 82, no. 1, pp. 41-53, Jan. 1994.
[24] P. Thambidurai and Y.-K. Park, "Interactive Consistency with Multiple Failure Modes," *Proc. Symp. Reliable Distributed Systems*, pp. 93-100, Columbus, Ohio, Oct. 1988.
[25] J. Turek and D. Shasha, "The Many Faces of Consensus in Distributed Systems," *Computer,* vol. 25, no. 6, pp. 8-17, June 1992.
[26] K.Q. Yan and Y.H. Chin, "An Optimal Solution for Consensus Problem in an Unreliable Communication System," *Proc. Int'l Conf. Parallel Processing*, pp. 388-391, University Park, Pa., Aug. 1988.
[27] K.Q. Yan, Y.H. Chin, and S.C. Wang, "Optimal Agreement Protocol in Byzantine Faulty Processors and Faulty Links," *IEEE Trans. Knowledge and Data Eng.,* vol. 4, no. 3, pp. 266-280, June 1992.

Hin-Sing Siu received the BM degree from Fu-Jen University, and the MS and PhD degrees in computer and information science from National Chiao-Tung University. Currently, he is an associate professor with the Department of Industrial Engineering and Management, MingChi Institute of Technology, Taipei, Taiwan. His research interests include fault tolerant distributed systems, computer networking, and database management systems.

Yeh-Hao Chin (S'69-M'72-SM'95) received the BSEE degree from National Taiwan University, and the MS and PhD degrees in electrical engineering from the University of Texas at Austin in 1970 and 1972, respectively. He has been a member of the faculty at Northwestern University, Cleveland State University, and National Chiao-Tung University. He has also worked for the Control Data Corporation, Sunnyvale, California, and AT&T Bell Laboratories, Holmdel, New Jersey. Currently, he is a professor with the Institute of Computer Sciences, National Tsing-Hua University, Hsinchu, Taiwan. His research interests include database management systems, operation systems, design and analysis of algorithms, software engineering, and VLSI design. He is a senior member of the IEEE Computer Society.

Wei-Pang Yang received the BS degree in mathematics from National Taiwan Normal University in 1974, and the MS and PhD degrees from National Chiao-Tung University in 1979 and 1984, respectively, both in computer engineering. Since August 1979, he has been on the faculty of the Department of Computer Science and Information Engineering at National Chiao-Tung University, Hsinchu, Taiwan. In the academic year 1985-1986, he was awarded the National Postdoctoral Research Fellowship and was a visiting scholar at Harvard University. From 1986 to 1987, he was the director of the Computer Center of National Chiao-Tung University. In August 1988, he joined the Department of Computer and Information Science at National Chiao-Tung University, and acted as the head of the department for one year. Then, he went to IBM Almaden Research Center in San Jose, California, for another a year as a visiting scientist. From 1990 to 1992, he was the head of the Department of Computer and Information Science again. His research interests include database theory, database security, object-oriented database, image database, and Chinese database retrieval systems.

Dr. Yang is a senior member of the IEEE Computer Society and a member of the ACM. He was the winner of the 1988 and 1992 AceR Long Term Award for Outstanding MS Thesis Supervision, the 1993 AceR Long Term Award for Outstanding PhD Dissertation Supervision, and the winner of the 1990 Outstanding Paper Award of the Computer Society of the Republic of China. He also received the Outstanding Research Award of National Science Council of the Republic of China.