Workshop on Computer Networks

Reaching Fault Detection Agreement

K.Q. Yan
Department of Business Administration
kqyan@mail.cyut.edu.tw

S.C. Wang
Department of Information Management
scwang@cyut.edu.tw

C.F. Cheng
Department of Information and Communication of Engineering
s9027601@mail.cyut.edu.tw

Chaoyang University of Technology
168 Gifeng E. Rd., Wufeng,
Taichung County, Taiwan 413, R.O.C.
TEL: 886-4-2332-3000 ext. 3071
Fax: 886-4-2374-2319

ABSTRACT

Siu, Chin and Yang proposed the protocol GPBA to solve the Byzantine Agreement (BA) problem in the presence of mixed faults on the processors and links in the general network. Subsequently, they proposed the FDAMIX protocol to solve the Fault Diagnosis Agreement (FDA) with mixed faults on the processors. However, they could not solve the FDA problem with mixed faults on the links. In this study, we shall propose a new protocol FDAL to solve the FDA problem with mixed faults on the links. That is, FDAL can detect/locate the faulty links to reconfigure the unreliable general network into a reliable general network and increase the performance, making sure of the integrity of the network.

Keywords: Byzantine agreement, fault diagnosis agreement, fault-tolerance, distributed system, mixed fault model.

I. INTRODUCTION

The reliability of the distributed system has become more and more important nowadays as a result of the growth of the Internet. The Byzantine Agreement (BA) [2,4,5,6,7,10,11,12,13,14] is one of the most important problems in designing a fault-tolerance distributed system. Under many circumstances, a fault-free processor in a distributed system can reach an agreement before performing some unique tasks [1,2,3]. For example, a well-known form of the problem is the transaction commit problem [3]. The problem is for all the data manager processors that participate in the processing of a particular transaction to agree on whether to record the results of transaction in the database or to discard them.

Another related problem is the Fault Diagnosis Agreement (FDA) problem [5,12]. The FDA is used to make each fault-free processor detect/locate all faulty components in the network. That is, if the FDA can be achieved, then each fault-free processor can identify all the faulty components in the network and ignore the influence from faulty components. So, the performance and integrity of a distributed system can be guaranteed.

In many previous studies [11], the BA problem has been visited in a general network [5,7,10,11] in the mixed fault model [5,7,10,11] (namely both arbitrary faults and dormant faults are present) on processors and links. Subsequently, the protocol FDAMIX [5] was proposed to solve the FDA problem with mixed faults on the processors. However, it cannot solve the FDA problem in the general network with mixed faults on the links.

The reliability of the connection state of the network is also an important topic in designing the distributed system because, if the connection state of the network is stable, we can transmit the message correctly and on time without faulty influences. On the contrary, if the connection state of the network is not reliable, then the message can possibly be influenced by the faulty links, getting changed and being not able to arrive on time.

The symptoms of faults on the links can be classified into three categories: crash faults, stuck-at faults, and arbitrary faults (also called Byzantine faults) [14]. A crash fault takes place when a link is broken. A stuck-at fault happens when the message received from a certain link is always a constant value. Finally, a link with the arbitrary fault is one whose behavior is unrestricted and arbitrary, so it causes the worst problem. A fault-free link can transmit messages on time and correctly, but the message which is transmitted by a faulty link may be changed or delayed. Fault-free processors can easily detect crash and omission faults if

the protocol appropriately encodes a transmitted message by either the Non-Return-to-Zero code or the Manchester code [9,14] before transmission, so we call them dormant faults. However, with arbitrary faults, things can by no accounts be so easy.

In this study, we shall propose two new protocols, the Fault Diagnosis Agreement on Link (DFAL) and the Virtual Relay Fault-tolerance Channel (VRFC), to solve the FDA problem with mixed faults on the links if it meets the following constraints:

(Agreement): ***All*** the fault-free processors identify the common set of faulty links in the process of reaching consensus.

(Fairness): No faulty link is falsely detected as fault-free by any fault-free processor, and no fault-free link is falsely detected as faulty by any fault-free processor.

The rest of this paper is organized as follows. Section Ⅱ will provide the detailed descriptions of our new protocols DFAL and VRFC. Then, in Section Ⅲ, we shall give an example of executing DFAL and VRFC. Section Ⅳ will serve to analyze the correctness of our protocols. Finally, in Section Ⅴ, we shall draw our conclusion and discuss our future work.

II. THE PROPOSED PROTOCLS

In this section, the proposed protocols FDAL and VRFC will be introduced to solve the FDA problem with mixed faults on the links in a general network. The assumptions and parameters of our protocols are listed as follows:

- The processors of the underlying network are assumed to be fault-free. (This can be achieved by using protocol FDAMIX [5].)
- Let $\boldsymbol{P}$ be the set of all the processors in the general network, and $|\boldsymbol{P}|=n$.
- Each processor in the network can be unique identified.
- Each processor in the general network has the same initial value. (This can be achieved by using protocol GPBA [11].)
- Let L_a be the maximum number of arbitrary faulty links in the general network.
- Let L_d be the maximum number of dormant faulty links in the general network.

- Let c be the connectivity of the general network, where $c > 2L_a + L_d$.
- A processor does not know the fault status of the links in the general network, but the dormant faulty links can be detected [9,14].

DFAL can detect/locate L_a arbitrary faulty links and L_d dormant faulty links in the general network, where $c > 2L_a + L_d$. Before executing DFAL, we must execute GPBA and FDAMIX first. Because, after executing GPBA and FDAMIX, we can ensure that the general network does not have arbitrary faulty processors and dormant faulty processors, and each fault-free processor in the general network can reach a common value. The commonly agreed value by GPBA can be used as the initial value of each fault-free processor in the general network for executing DFAL.

There are four phases in our protocol DFAL, and they are the message exchange phase, the fault diagnosis phase, the result exchange phase, and the reconfigure phase. The number of rounds of message exchange needed is only two (one round for transmitting the initial value and the other round for transmitting the fault diagnosis report). The detailed definition of proposed protocol DFAL is shown in Figure 1.

2.1 The Message Exchange Phase

In the message exchange phase, each processor P_i has the same initial value from GPBA. Then each processor P_i transmits the initial value v_i to all the other processors through links and receives the value v_j from processor P_j, for $1 \le i,j \le n$, to construct the vector $V_i = [v_1, v_2, \ldots, v_j, \ldots, v_n]$. If no connected link, say il, can be found, then $v_l = \varphi$. If a dormant faulty link, say ik, is found, then $v_k = \lambda$. After message exchange in the message exchange phase, each processor in the general network can receive the partial messages because the network is not fully connected. As a result, most of the processors cannot get the whole messages from all the other processors.

2.2 The Fault Diagnosis Phase

In the fault diagnosis phase, each processor P_i searches each value in the vector V_i, where $1 \le i \le n$. If the value v_k in the vector V_i is λ, then the link between ik is a dormant faulty link, for $1 \le k \le n$. Therefore, by step 1 in the fault diagnosis phase, we can detect/locate the dormant faulty link. Then, search each value in the

vector V_i, where $1 \leq i \leq n$. If the value v_k in the vector V_i is not the agreed-upon value from GPBA and not λ or φ either, then the link between *ik* is an arbitrary faulty link, for $1 \leq k \leq n$. By step 2 in the fault diagnosis phase, we can detect/locate the arbitrary faulty link. Due to the fact that the network is not fully connected, we need the third phase, namely the result exchange phase, to get the Link Fault Diagnosis Report (LFDR) in unanimity.

2.3 The Result Exchange Phase

In the result exchange phase, we need to introduce a modified transmitting protocol, our Virtual Relay Fault-tolerance Channel (VRFC) inspired by the concept of virtual link by F.J. Meyer and D.K. Pradhan [7]. Using VRFC, we can make an un-fully connected network work just like a fully connected network. The detailed definition of VRFC is shown in Figure 2.

For example, in Figure 3(a), there is a general network with six processors, and the connectivity of the network is four. The connection state of processor P_1 and processor P_2 is shown in Figure 3(b). If processor P_1 wants to transmit a message to processor P_2, it can match the node-disjoint path as shown in Figure 3(c) by node-disjoint rule [8] (each intermediate processor on these *c* paths should not be passed through more than once). So, each receiver processor can receive four messages from the sender processor. In Figure 3(c), the link between processor P_2 and processor P_3 is in arbitrary fault, and the link between processor P_2 and processor P_4 is in dormant fault. That is, there are two values that may not be correct, but we can ignore the faulty influence by the concept of majority if $c>2L_a+L_d$. That is, we can make sure that using VRFC to transmit message can provide reliable communication without any influence from faulty links and thus that the message can arrive accurately if $c>2L_a+L_d$.

Therefore, each processor P_i produces $VRFC\_Message_i$ by step 1 and step 2 in the fault diagnosis phase. The form of VRFC_Message is shown in Figure 1. Then, each processor P_i uses VRFC to transmit $VRFC\_Message_i$ to all the processors in the general network. In the step 2 of the result exchange phase, each processor constructs T_VRFC_Messages out of $VRFC\_Message_j$ sent by each free-fault processor P_j for $1 \leq j \leq n$. Finally, T_VRFC_Messages can be used to produce the LFDR. The form of VRFC_Message is shown in Figure 1.

2.4 The Reconfigure Phase

We can use the LFDR to reconfigure the network. After the reconfiguration, the performance and integrity of the network can be guaranteed.

Protocol DFAL (For all fault-free processor P_i with the same initial value, $1 \le i \le n$)

Message Exchange Phase:

Transmit the agreed-upon value v_i from GPBA [11] to all the other processors and receive the value v_j from processor P_j, for $1 \le i,j \le n$. Then, construct the vector $V_i = [v_1, v_2, \ldots, v_j, \ldots, v_n]$, $1 \le j \le n$. If there is no connected link found, say *il*, then $v_l = \varphi$. If a dormant faulty link, say *ik*, is found, then $v_k = \lambda$.

Fault Diagnosis Phase:

Step 1: Search each value in the vector V_i, where $1 \le i \le n$. If the value v_k in the vector V_i is λ, then the link between *ik* is a link in dormant fault, for $1 \le k \le n$.

Step 2: Search each value in the vector V_i, where $1 \le i \le n$. If the value v_k in the vector V_i is not the agreed-upon value from GPBA and not λ or φ either, then the link between *ik* is a link in arbitrary fault, for $1 \le k \le n$.

Result Exchange Phase:

Step 1: By step 1 and step 2 in the fault diagnosis phase, each processor P_i can produce the $VRFC\_Message_i$. Then each processor P_i transmits the $VRFC\_Message_i$ by step1 and step 2 by using VRFC.

Step 2: Each processor constructs the T_VRFC_Messages out of the $VRFC\_Message_j$ sent by each free-fault processor P_j for $1 \le j \le n$.

Step 3: Use the T_VRFC_Messages to produce the Link Fault Diagnosis Report (LFDR).

Reconfigure Phase:

Step1: According to the LFDR, reconfigure the network.

The form of "VRFC_Message": {Q_a, ALink_ID, Q_d, DLink_ID}

Q_a: the number of arbitrary faulty links.
Q_d: the number of dormant faulty links.
ALink_ID: the arbitrary faulty link identification.
DLink_ID: the dormant faulty link identification.

The form of "LFDR":

Faulty type	Quantity	Link_ID
Arbitrary faulty Links		
Dormant faulty Links		

Figure 1. The proposed DFAL protocol

Protocol VRFC (Virtual Relay Fault-tolerance Channel)

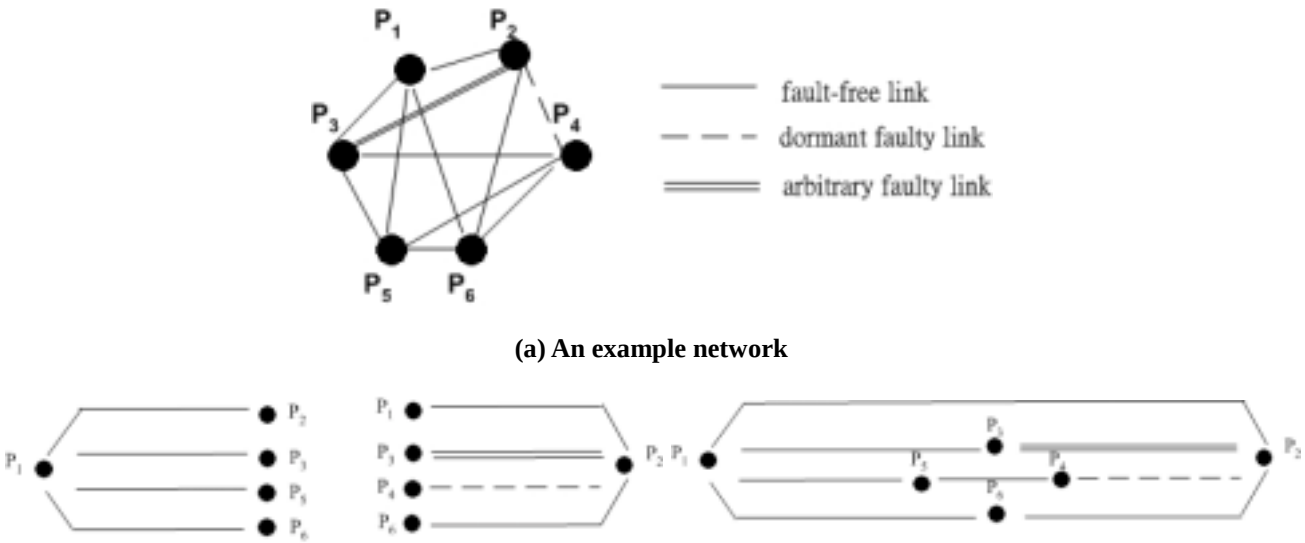
Definition:

- Each processor has the common knowledge of Graphic information G=(E,P), where P is the set of processors in the network and E is a set of processor pairs, (P_i,P_j), indicating a link between processor P_i and processor P_j ,where $1 \le i,j \le n$
- There are c ($c>2L_a+L_d$) paths from sender processor to receiver processor.
- These c paths from sender processor to receiver processor are node-disjoint paths.
- Each intermediate processor on these c paths should not be passed through more than once.

Steps:

1. Each sender processor P_i ($1 \le i \le n$) transmits the VRFC_Message$_i$ to each receiver processor P_j ($1 \le j \le n$) through c node-disjoint paths.
2. The receiver processor receives c VRFC_Message$_i$ from the same sender processor.
3. The receiver processor takes the majority message of VRFC_Message$_i$ to construct the T_VRFC_Messages.

Figure 2. The proposed VRFC protocol



(a) An example network

(b) The connection state of the sender processor P_1 and that of the receiver processor P_2

(c) The node-disjoint paths from sender processor P_1 to receiver processor P_2

Figure 3. An example of node-disjoint paths

III. An example of executing VRFC & DFAL

Here is an example of executing our DFAL and VRFC. A general network with fault-free processors by using FDAMIX is in Figure 4(a). There are six processors in the network, and the connectivity of the network is four. The arbitrary faulty link is between processor P_1 and processor P_2. The dormant faulty link is between processor P_2 and processor P_5.

The initial value of each fault-free processor is the agreed-upon value (the agreed-upon value is

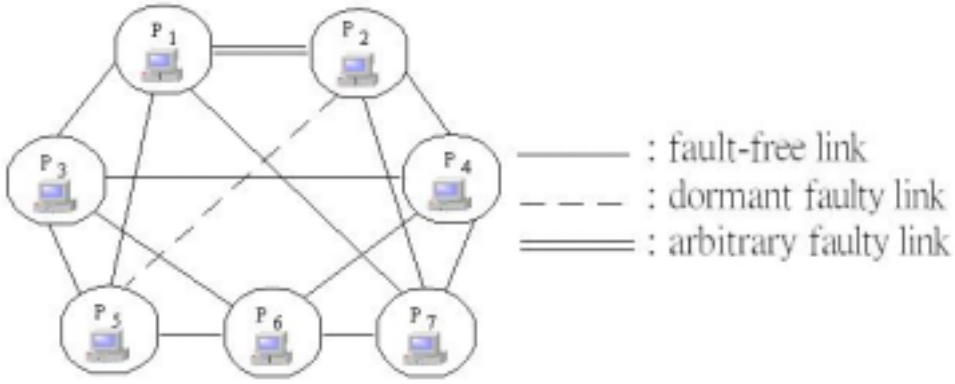
assumed as 1) from GPBA as illustrated in Figure 4(b).

In the messages exchange phase, each processor P_i transmits its initial value v_i to all the other processors through the connected links in the first round, for $1 \le i \le n$. The messages received by processors P_1, P_2, P_3, P_4, P_5, P_6 and processor P_7 in the first round are illustrated in Figure 4(c).

In the fault diagnosis phase, by step 1 and step2, each processor is only able to locate/detect the partial dormant faulty links and arbitrary links as shown in Figure 4(d) because the network is not fully connected.

In the result exchange phase, each processor P_i uses VRFC to transmit the VRFC_Message$_i$ from the fault diagnosis phase to all the processors. Then, each processor uses the received messages to create the T_VRFC_Messages as shown in Figure 4(e). Then, by using the T_VRFC_Messages, each processor produces the link fault diagnosis report as shown in Figure 4(f). Each processor produces the same LFDR by using VRFC and DFAL.

In the reconfigure phase, from LFDR, each processor can obtain the facts that the link between processor P_1 and P_2 is in arbitrary fault and that the link between processor P_2 and P_5 is in dormant fault. Each processor can reconfigure the network by eliminating the link between processor P_1 and P_2 and the link between processor P_2 and P_5. The network after reconfiguration is shown in Figure 4(g).



(a) A network with six processors and the connectivity of four.

V_1	1
V_2	1
V_3	1
V_4	1
V_5	1
V_6	1
V_7	1

(b) Initial value

Figure 4. An example of executing DFAL and VRFC (Cont'd)

V_1	V_2	V_3	V_4	V_5	V_6	V_7
1	0	1	φ	1	φ	1
0	1	φ	1	λ	φ	1
1	φ	1	1	1	1	φ
φ	1	1	1	φ	1	1
1	λ	1	φ	1	1	φ
φ	φ	1	1	1	1	1
1	1	φ	1	φ	1	1

(c) Vectors received after the first round

VRFC_P_1	{1, Link_1-2, 0, Null}
	The VRFC_Message$_1$ sent by P_1
VRFC_P_2	{1, Link_1-2, 1, Link_2-5}
	The VRFC_Message$_2$ sent by P_2
VRFC_P_3	{0, Null, 0, Null}
	The VRFC_Message$_3$ sent by P_3
VRFC_P_4	{0, Null, 0, Null,}
	The VRFC_Message$_4$ sent by P_4
VRFC_P_5	{0, Null, 1, Link_2-5}
	The VRFC_Message$_5$ sent by P_5
VRFC_P_6	{0, Null, 0, Null}
	The VRFC_Message$_6$ sent by P_6
VRFC_P_7	{0, Null, 0, Null}
	The VRFC_Message$_7$ sent by P_7

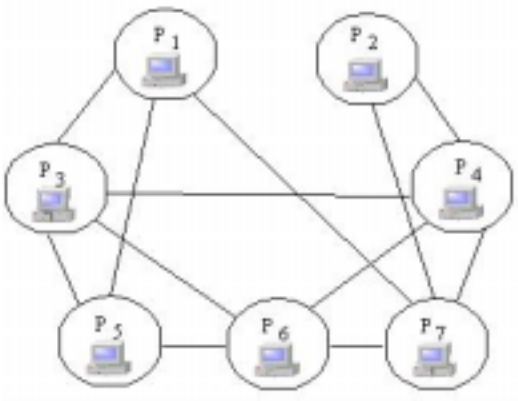
(d) The VRFC_Message sent by each processor

VRFC_P_1	{1, Link_1-2, 0, Null}
VRFC_P_2	{1, Link_1-2, 1, Link_2-5}
VRFC_P_3	{0, Null, 0, Null}
VRFC_P_4	{0, Null, 0, Null,}
VRFC_P_5	{0, Null, 1, Link_2-5}
VRFC_P_6	{0, Null, 0, Null}
VRFC_P_7	{0, Null, 0, Null}

(e) The T_VRFC_Messages of each processor P_i by VRFC

Faulty type	Quantity	Link_ID
Arbitrary faulty Links	1	Link_1-2
Dormant faulty Links	1	Link_2-5

(f) Using the T_VRFC_Messages to produce the Link Fault Diagnosis Report (LFDR)



(g) The network without faulty links after reconfiguration

Figure 4. An example of executing DFAL.

IV. THE CORRECTNESS OF PROPOSED PROTOCOLS

The following lemmas and theorems are used to prove the correctness of DFAL. It can detect/locate L_a arbitrary faulty links and L_d dormant faulty links in the general network, where $c > 2L_a + L_d$.

Theorem 1: Any fault-free processor P_i can detect dormant faulty links which connected to the fault-free processor P_i, where $1 \le i \le n$.

Proof: The fault-free processor can detect dormant faults if the protocol appropriately encodes a transmitted message by either the Non-Return-to-Zero code or the Manchester code [9,14] before transmission.

Lemma 1: Each fault-free processor's initial value is the same.

Proof: After executing GPBA [11], each fault-free processor in the general network has the same initial value well agreed upon.

Lemma 2: If the link between sender processor P_i and receiver processor P_j is fault-free, then the received value v_{ij} is the same as the initial value.

Proof: If the link between sender processor P_i and receiver processor P_j is fault-free, then the message sent through the link is transmitted correctly. That is, v_i is equal to v_{ij}.

Theorem 2: Any fault-free processor P_i can detect arbitrary faulty links which connected to the fault-free processor P_i, where $1 \le i \le n$.

Proof: If the value v_{ij} is not the initial value and not λ or φ either, then the link between processors I and j is in arbitrary fault, By Lemma 1 and Lemma 2, each fault-free receiver processor must receive the same value from other fault-free processors.

Lemma 3: Any fault-free processor P_i can detect/locate the partial faulty links to processor P_i in the fault diagnosis phase in a general network.

Proof: Due to the network topology not fully connected, a fault-free processor P_i can only detect/locate the partial faulty links connected to processor P_i in the fault diagnosis phase.

Lemma 4: Fault-free receiver processor can receive the messages VRFC_Message$_i$ from fault-free processor P_i, for $1 \le i \le n$, without being influenced by faulty links, if $c > 2L_a + L_d$.

Proof: The fault-free processor P_i sends c copies of VRFC_Message$_i$ to the fault-free receiver processor. In the worst case, the fault-free receiver processor can receive $c - L_d$ messages transmitted by the

fault-free processor P_i because dormant fault components can be detected. Since $c - L_d > 2L_a$, the fault-free receiver processor can decide whether the message was from processor P_i or not by taking the majority value. Then, construct T_VRFC_Messages by the majority value from VRFC_Message$_i$.

Theorem 3: After the result exchange phase, the LFDR of each fault-free processor is the same.

Proof: After the result exchange phase, each fault-free processor receives whole messages from all the other processors. By Lemma 3 and Lemma 4, each fault-free processor can construct the same LFDR in the general network.

Theorem 4: The DFAL protocol can detect/locate the faulty links in a general network.

Proof: By theorem 1, theorem 2 and theorem 3, DFAL can detect/locate the faulty links in a general network.

V. CONCLUSION

Due to the recent popularity of distributed systems, the reliability of the distributed system has become a more and more important topic of researches. At the same time, fault diagnosis has also become an attention-drawing topic. In [11] by Siu et al., they proposed the GPBA protocol to solve the BA problem in the general network with mixed faults on both processors and links. And then, they also proposed another protocol, FDAMIX, to solve the FDA problem in the general network with mixed faults on the processors. However, they have not solved the FDA problem in the general network with mixed faults on the links. In this study, the proposed protocol FDAL proves to be able to solve the FDA problem with mixed faults on the links. That is, by using GPBA, FDAMIX, and FDAL together, we can solve the BA problem and FDA problem with mixed faults on both processors and links in a general network.

After reaching the common agreement and fault diagnosis in a general network, we can reconfigure the network and eliminate the faulty processors and faulty links to enhance the performance and strengthen the integrity of the network. This is of special importance to high reliability applications such as a life-critical distributed system.

In short, the proposed protocols FDAL and VRFC only consume a minimum number of rounds of message exchange and detect/locate a maximum number of links with mixed faults in a general network.

Although GPBA, FDAMIX, and FDAL are designed to deal with processor failures and link failures in a general network, they cannot be used in the Multi-Casting Network (MCN), which is a more reliable and applicable network topology that allows a frame transferring to an individual, a broadcast, or a group address

in a LAN, where the routers help local processors monitor the messages on the Internet. Our future work will be focused on the network topology of the MCN.

REFERENCES

[1] A. Bar-Noy et al., "Shifting Gears: Changing Algorithms on the Fly To Expedite Byzantine Agreement," *Proc. Symposium on Principles of Distributed Computing*, pp. 42-51, 1987.

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

[3] Skeen, D., and Stonebraker, M. "A Formal Model of Crash Recovery in a Distributed System, " *IEEE Trans. Software Engineering*, vol. 9, no 4, pp. 219-228, May 1983,

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

[5] H.S. Hsiao, Y.H. Chin, W.P. Yang, "Reaching Fault Diagnosis Agreement under a Hybrid Fault Model," *IEEE Trans. Computers*, vol. 49, no. 9, pp. 980-986, September 2000.

[6] 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.

[7] 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, April 1991.

[8] Deo, Narsingh, Graph Theory with Applications to Engineering and Computer Science, Englewood Cliffs, N. J.:Prentice-Hall, 1974.

[9] F. Halsall, Data Links, Computer Networks and Open Systems, 4th ed., Addison-Wesley, Reading, MA, 1995.

[10] H.S. Siu, Y.H. Chin, and W.P. Yang, "A Note on Consensus on Dual Failure Modes," *IEEE Trans. Parallel and Distributed System*, vol. 7, no. 3, pp. 225-230, March 1996.

[11] H.S. Siu, Y.H. Chin, and W.P. Yang, "Byzantine Agreement in the Presence of Mixed Faults on Processors and Links," *IEEE Trans. Parallel and Distributed System*, vol. 9, no. 4, pp. 335-345, April 1998.

[12] S.C. Wang, Y.H. Chin, and K.Q. Yan, "Reaching a Fault Detection Agreement," *Proc. Int'l Conf. Parallel Processing*, 1990, pp. 251-258.

[13] S. C. Wang, Y. H. Chin, and K. Q. Yan, “Byzantine Agreement in a Generalized Connected Network Model,” *IEEE Trans. Parallel and Distributed System*, 6(4), pp. 420-427, 1995.

[14] K.Q. Yan, S.C. Wang and Y.H. Chin, “Consensus Under Unreliable Transmission,” *Information Processing Letters*, vol. 69, pp.243-248, March 1999.