國立交通大學

資訊科學與工程研究所

博士論文

混合式網路下協議問題之研究

Agreement Problems under Combined Wired/Wireless Network

研 究 生：鄭建富

指導教授：梁　婷 博士

　　　　　王淑卿 博士

中華民國九十七年九月

混合式網路下協議問題之研究

Agreement Problems under Combined Wired/Wireless Network

研究生：鄭建富　　Student : Chien-Fu Cheng

指導教授：梁婷 博士　　Advisor : Dr. Tyne Liang

王淑卿 博士　　Dr. Shu-Ching Wang

國立交通大學

資訊科學與工程研究所

博士論文

A Dissertation Submitted to
Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy
in
Computer Science

September 2008
Hsinchu, Taiwan, Republic of China

中華民國九十七年九月

混合式網路下協議問題之研究

學生：鄭建富　　　　　　　　　　指導教授：梁　婷 博士

王淑卿 博士

國立交通大學 資訊科學與工程研究所

摘　　要

近年來由於無線通訊以及行動計算的蓬勃發展使得現今的網路型態走向了混合式的網路（包含了有線網路及無線網路）。因此混合式網路的可靠度以及容錯問題成為非常重要的課題。 為了提供可靠的計算環境，我們需要一個機制讓參與運作的處理器即使在發生錯誤或攻擊時仍能夠達成協議。 因此拜占庭協議問題以及合議問題成為許多學者們關注的焦點。 然而，過往的拜占庭協議問題以及合議問題的研究大都是針對在有線的網路之上。由於有線網路的實體拓樸為靜態的，使得過往針對這些固定式網路所提出的協定無法運作在混合式網路之中，其主要原因在於混合式網路之實體拓樸為動態的。

在本論文中，我們將探討拜占庭協議問題以及合議問題於混合式網路。為了因應混合式網路的特性（行動式處理器的運算能力通常較固定式處理器要來的弱）以及減少為了達成協議所需要的訊息交換次數，因此在我們所設計的協定之中導入了階層式的概念，將大部分的通訊工作以及運算工作交由伺服器來處理，如此一來將可以大幅降低行動式處理器的負擔。以效能的觀點來看，本論文中所提出的協定使用了最少的訊息交換次數並且可以容忍最大數量的損毀處理器。此外，由於純粹的有線網路以及純粹的無線網路都是混合式網路的特例，因此我們也將討論協議問題於這兩種網路之上。

為了提供更高可靠度的計算環境,我們還必須偵測/定位出網路中的損毀處理器。 因此，我們也將提出一個新的錯誤診斷協定來偵測/定位出損毀的處理器於混合式網路之中。a

在應用方面，近年來檔案分享已經成為點對點網路上最受歡迎的一項服務，而許多的點對點網路都是架構在有線及無線的混合式網路之上。這也使得檔案一致性的問題成為一項非常重要的課題。 因為惡質性損毀的攻擊者可能會隨意變更檔案的內容，並且將這些更改過的檔案傳送給其他的處理器。如此一來將使網路上充斥著不一致的檔案，造成浪費許多的資源，例如：頻寬、儲存空間以及傳輸時間。 因此確認所擁有的檔案是否為正確是相當重要的。 在本研究當中我們將使用所提出的協定來解決檔案一致性的問題於點對點網路之中。

關鍵字：*容錯分散式系統; 拜占庭協議問題; 合議問題; 錯誤偵斷問題; 惡質性損毀; 非惡質性損毀; 動態網路; 有線及無線混合式網路; 點對點網路; 檔案分享*

Agreement Problems under Combined Wired/Wireless Network

Student: Chien-Fu Cheng

Advisor: Dr. Tyne Liang
Dr. Shu-Ching Wang

Institute of Computer Science and Engineering
National Chiao Tung University

ABSTRACT

Since wireless communication and mobile computing are becoming more and more ubiquitous, most network environments today are combined wired and wireless. The reliability and fault tolerance of the combined wired/wireless network has become an important topic. In order to provide a reliable environment, a mechanism that allows a set of processors to reach a common agreement, even in the presence of faulty processors, is needed. Therefore, the Byzantine Agreement (BA) problem and Consensus problem have drawn attention of more researchers. Traditionally, most of the BA problem and Consensus problem were focused on wired networks. We know that the physical topology of a wired network is static, but the physical topology of a combined wired/wireless network is dynamic. Thus, previous BA and Consensus protocols for static network are not applicable in a combined wired/wireless network.

In this dissertation, we visited the BA problem and Consensus problem in combined wired/wireless network. In order to meet the characteristics of combine wired/wireless networks (the limited resources have made the computation ability of mobile processors often weaker than that of stationary processors) and reduce the number of rounds of message exchange required, most of the communications and computation overhead must be fulfilled within by the servers. Therefore, we introduce a hierarchical concept in our system model. Only servers need to exchange messages and compute the common value. From the performance perspective, the proposed protocols use the minimum number of message exchanges and can tolerate the maximum number of faulty processors allowed in the networks. AS a matter of fact, pure wired networks and pure wireless networks are all special case of the combined wired/wireless networks. We also discuss the BA and Consensus problems in pure wired network and pure wireless network.

In a highly reliable fault-tolerant environment, to reach a common agreement is not enough. It is also necessary to detect/locate the faulty components in the network. Therefore, we also propose a new protocol to solve the Fault Diagnosis Agreement (FDA) problem in combined wired/wireless networks.

In the usage, the file-sharing application has been the most popular application in Peer-to-Peer (P2P) systems. Many P2P networks are overlay networks because they run on top of the combined wired/wireless network. Hence, the Consensus problem of file-sharing has become an important topic. As malicious attackers may modify files arbitrarily and spread inconsistent files to other processors in the P2P network, inconsistent files will not only spread in P2P networks but also waste resources, such as bandwidth, space of storage and transmission time. Hence, how to make fault-free processors ensure that the files they hold are correct is an import topic. In this thesis, we give an application of Consensus protocol to ensure the file consistency of file-sharing in P2P systems.

Keywords: *fault-tolerant distributed system, Byzantine agreement problem, Consensus problem, fault diagnosis agreement problem, malicious fault, dormant fault, dynamic network, combined wired/wireless network, Peer-to-Peer system; file-sharing.*

ACKNOWLEDGEMENT

（誌謝）

博士論文能夠完成，首先要感謝的是指導教授梁婷博士以及共同指導教授王淑卿博士，感謝兩位老師在我研究所生涯中對我學術研究的指導以及啟發。兩位指導老師嚴謹的治學態度，讓我在學術研究上學習到謹慎的思考，也在日常生活上獲益良多。 此外也感謝系上的楊武教授、蔡錫鈞教授以及譚建民教授在計畫書口試、校內口試以及校外口試時所提供的許多寶貴建議。

感謝校外口試委員：王有禮教授、石維寬教授、金仲達教授、徐力行教授在口試過程中提供許多寶貴的建議，讓我的論文能更趨完善。感謝實驗室的同仁們，你們是我博士班研究生涯的好伙伴，再次謝謝你們在這段過程中對我的幫忙以及鼓勵。

我要把我最深的感謝留給我的家人，因為有你們的支持與鼓勵，讓我在求學的過程中有足夠的動力去完成這個學位，謝謝你們。

要感謝的人太多，僅以此論文獻給我最愛的親人、師長與朋友。

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF NOTATIONS

AES	Advanced Encryption Standard
AFDA	Adaptive Fault Diagnosis Agreement Protocol
BA	Byzantine Agreement
CCP	Client-initiated Consensus Protocol
c	*c* is the connectivity of each server in the higher network level of combined wired/wireless network
c_d	c_d is the maximum number of dormant faulty communication links allowed in the higher network level of combined wired/wireless network
c_m	c_m is the maximum number of malicious faulty communication links allowed in the higher network level of combined wired/wireless network
CP$_{p2p}$	Consensus Protocol for P2P network
EAP	Eventual Agreement Problem
DES	Data Encryption Standard
DHT	Distributed Hash Table
FDA	Fault Diagnosis Agreement
IAP	Immediate Agreement Problem
ic-tree	Information Collecting Tree, the information collecting tree is a tree structure used to store the messages and to remove the influence from the repeated processors
IC-trees	***IC-trees*** =[ic-tree$_{s,}$ ic-tree$_a$, ic-tree$_b$ …, ic-tree$_{\ddot{v}}$] , where $\ddot{v}$ is the last Processor id in the network by alphabetical order. ***IC-trees*** is the set of processors' ic-tree.
MAHAP	Mobile Ad-Hoc Agreement Protocol
MANET	Mobile Ad-hoc Network
mg-tree	Message Gathering Tree, the message gathering tree is a tree structure used to store the messages

N	N is the set of all processors in the network and $\lvert N \rvert = n$, where n is the number of processors in the underlying network, and $n \geq 4$.
P2P	Peer-to-Peer
p_a	p_a is the maximum number of *away processors* allowed in the underlying network
p_d	p_d is the maximum number of dormant faulty processors allowed in the underlying network
p_m	p_m is the maximum number of malicious faulty processors allowed in the underlying network
QoS	Quality of Service
SBAP	Server-initiated Byzantine Agreement Protocol
TTL	Time-To-Live
z_d	z_d is the maximum number of dormant faulty servers allowed in the combined wired/wireless network
z_m	z_m is the maximum number of malicious faulty servers allowed in the combined wired/wireless network, $z_m \leq \lfloor (z_n-1)/3 \rfloor$
Z_N	Z_N is the set of all servers in the combined wired/wireless network and $\lvert Z_N \rvert = z_n$, where z_n is the number of servers in the underlying network and $z_n \geq 4$
γ	γ is the number of rounds required
δ^0	The value δ^0 is used to report an absent value
λ^0	The value λ^0 is used to report a value from dormant faulty component
ϕ	ϕ is the default value

Chapter 1

Introduction

A distributed system is a collection of autonomous computers linked by a network, with software designed to produce an integrated computing facility. [12] . Since distributed computing is becoming more and more ubiquitous, the reliability and fault tolerance of the distributed system has become an important topic. For a system to be reliable, it is necessary to create a mechanism that allows a set of processors to agree on a common value [47] . With this agreement, several important distributed services can be achieved, such as clock synchronization, resource allocation, replicated file system [5] , atomic broadcast, atomic commitment and group membership [10] [21] [56] .

1.1 Problem Definition

Byzantine Agreement (BA) and Consensus problem are the most fundamental problems to reach a common agreement in distributed systems. The BA problem was first introduced by Pease, Shostak and Lamport [37] . In the BA problem, there are n ($n\geq4$) processors in a distributed system and an initial value v_s is set in a commander processor so that the commander processor can send its initial value to the other processors. The goal of a BA protocol is to make each fault-free processor reach a common agreement value. Protocols designed to deal with the BA problem should satisfy the following requirements:

(BA_Agreement): ***All*** fault-free processors agree on a common value;

(BA_Validity): If the source (commander) processor is fault-free, then ***all*** fault-free processors agree on the initial value that the source processor sends.

In the Consensus problem, each processor has its own initial value and sends its initial value to others. After the protocol is executed, each fault-free processor reaches a common agreement value. Protocols designed to deal with the Consensus problem should satisfy the following requirements:

(Consensus_Agreement): ***All*** fault-free processors agree on a common value;

(Consensus_Validity): If the initial value of all processors is v_i, then ***all*** fault-free processors shall agree on v_i.

In a highly reliable fault-tolerant distributed system, just reaching a common agreement is not enough. We need to take into consideration another related problem called the Fault Diagnosis Agreement (FDA) problem. The goal of solving the FDA problem is to make each fault-free processor detect/locate the common set of faulty components in the distributed system. After reaching the FDA, each fault-free processor can maintain the performance and integrity of the distributed system to provide a stable environment. Protocols designed to deal with the FDA problem should satisfy the following requirements:

(FDA_Agreement): ***All*** fault-free processors must identify the common set of faulty processors;

(FDA_Fairness): ***NO*** fault-free processor is incorrectly detected as faulty by any fault-free processor.

1.2 Motivation

In recent years, wireless networks and mobile computing are becoming ubiquitous. Most network environments today are combined wired and wireless. The combined wired/wireless networks have the advantages of both wired (e.g., powerful computation ability, high bandwidth, reliability, and so on.) and wireless networks (e.g., mobility, quick deployment, and so on). Previous BA and Consensus protocols for wired network [3] [28] [37] [51] [52]

[60] [61] [65] [70] were not applicable in combined wired/wireless networks. In this dissertation, the BA and Consensus problems in the combined wired/wireless network are re-examined. Moreover, we also propose a fault diagnosis agreement protocol to detect/locate the faulty processors. In the usage, many P2P networks are overlay networks because they run on top of the combined wired/wireless network. Hence, we also give an application of Consensus protocol to ensure the file consistency of file-sharing in P2P networks.

1.3 Organization of Dissertation

This dissertation consists of nine chapters, and the remainder is organized as follows. Chapter 2 describes the survey of related research works. Chapter 3 describes the basic concepts and approaches. Chapter 4 elaborates on the proposed BA protocol (Mobile Ad-Hoc Agreement Protocol, MAHAP) for wireless network. Chapter 5 provides detailed descriptions of the proposed BA protocol (Server-initiated Byzantine Agreement Protocol, SBAP) for combined wired/wireless network. Chapter 6 elaborates on the proposed Consensus protocol (Client-initiated Consensus Protocol, CCP) for combined wired/wireless Network. Chapter 7 provides detailed descriptions of the proposed FDA protocol (Adaptive Fault Diagnosis Agreement Protocol, AFDA). Chapter 8 gives an application of Consensus protocol (Consensus Protocol for P2P Network , CP_{p2p}) in file-sharing P2P system. Finally, conclusion and future work are given in Chapter 9.

Chapter 2

A Survey of Related Research Works

This chapter reviews the network structures, the types of fallible components and the related work of the BA, Consensus and FDA problems.

2.1 Network Structures

The network models can be classified into three categories based on their mobility features. The three types are pure wired network, pure wireless network and combined wired/wireless network.

2.1.1 Pure Wired Network

A wired network consists of a hard-wired backbone and powerful computing processors. Therefore, the bandwidth speed, computation ability and reliability of wired networks are generally much better than those of wireless networks. There are many kinds of pure wired network. They are basically classified into fully connected network [37] , broadcast network [3] , generalized connected network [60] , general network [51] and multicasting network [61] . A fully connected network is a mesh network in which each of the processors is connected to each other. An example of fully connected network is shown in Figure 2-1. A broadcast network is a network architecture in which a set of processors are connected via a shared communications line, called a bus. Figure 2-2 shows an example of broadcast network. A generalized connected network is a network structure that combines the fully connected network and broadcast network. It has the following features: (1) Grouping: A local bus links the processors of a same group. (2) Group member: The number of processors in each group is the same. (3) Connectivity: Each group is connected to each other. An example of generalized connected network is shown in Figure 2-3. A general network is an un-fully connected network. Figure 2-4 shows an example of general network. A multicasting network is a network structure that combines the general network and broadcast network together. The multicasting network has the following features: (1) Grouping: A local bus links the processors of a same group. (2) Group member: The number of processors in each group can be different from each other. (3) Connectivity: It allows the multicasting network a bounded connectivity *c*, where *c* is a constant. As a matter of fact, fully connected network, broadcast network, generalized connected network and general network are all special cases of multicasting network. An example of multicasting network is shown in Figure 2-5.

We know that the processors in a wired network do not have mobility. Hence, the physical topology of a wired network is static.

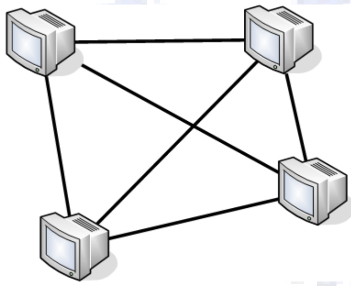


Figure 2-1. An example of fully connected network

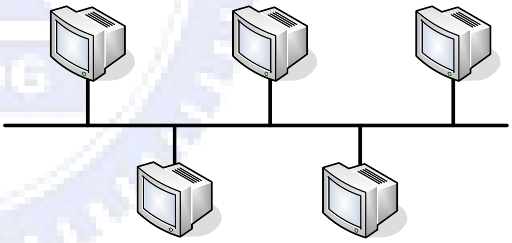


Figure 2-2. An example of broadcast network

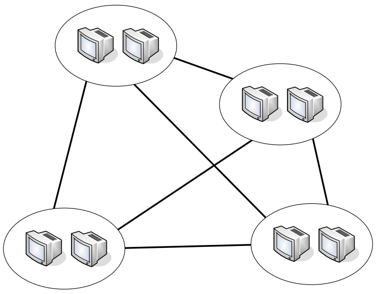


Figure 2-3. An example of generalized connected network

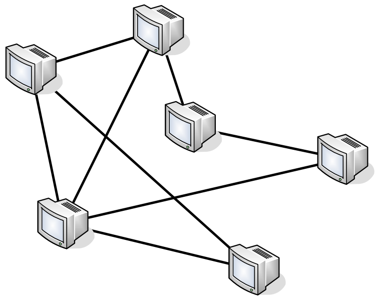


Figure 2-4. An example of general network

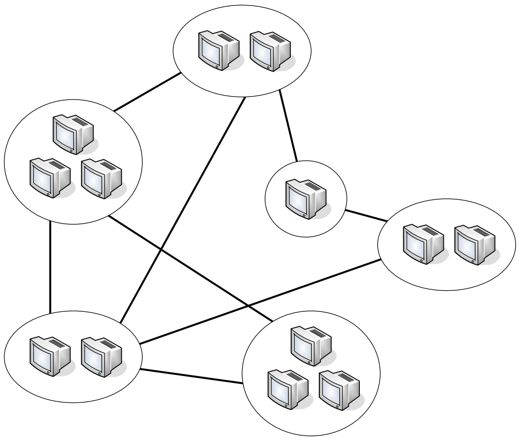


Figure 2-5. An example of multicasting network

2.1.2 Pure Wireless Network

In recent years, wireless network have become more popular. Mobile Ad-Hoc Network (MANET) is one type of non-fixed infrastructure wireless networks. Hence, MANETs have enjoyed an amazing rise in popularity. Because, the features of MANET are infrastructure less (no access point or base stations, no dedicated routers), automatic adaptation to changes in topology (nodes enter and leave the network freely, and mobility within the network) and quick deployment. Therefore, the MANETs are very attractive for tactical communications in the military, law enforcement, and conferences [22] . But, the limited resources (e.g., bandwidth and limited power) make the computation ability of mobile processors weaker than that of stationary processors. An example of wireless network is shown in Figure 2-6.

We know that the processors in a wireless network have mobility. Hence, the physical topology of a wireless network is dynamic.

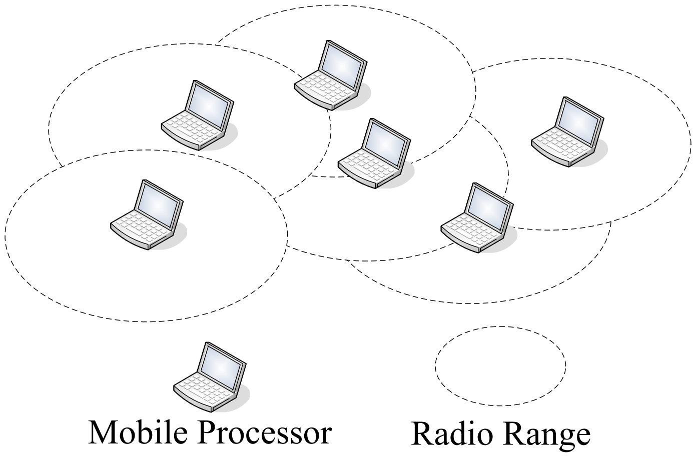


Figure 2-6. An example of wireless network

2.1.3 Combined Wired/Wireless Network

Since wireless networks and mobile computing are becoming ubiquitous, most network environments today are combined wired and wireless. The combined wired/wireless networks have the advantages of both wired (e.g., powerful computation ability, high bandwidth, reliability, and so on.) and wireless networks (e.g., mobility, quick deployment, and so on). An example of combined wired/wireless network is shown in Figure 2-7.

Some of processors have mobility in combined wired/wireless network, so the physical topology of combined wired/wireless network is also dynamic.

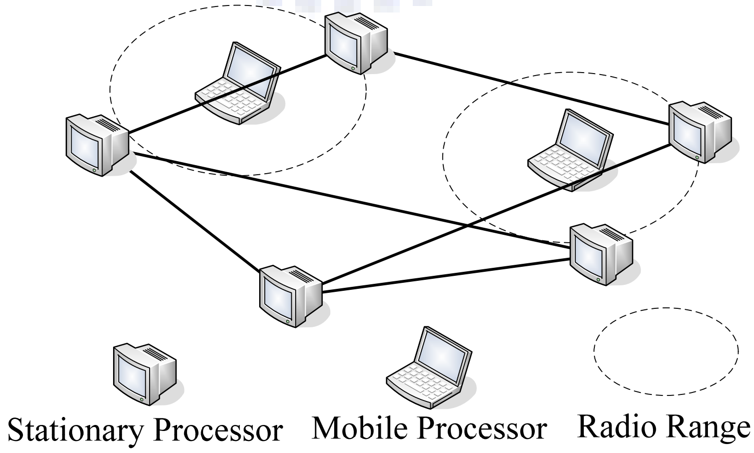


Figure 2-7. An example of combined wired/wireless network

2.2 The Fallible Components

Agreement may not take place for two reasons. Firstly, processors themselves may be faulty, resulting in unpredictable behaviors. Secondly, the communication link may be faulty, resulting in lost or garbled messages [47] .

2.2.1 The Symptoms of a Faulty Processor

During the execution of a protocol, a processor is said to be fault-free if it follows the protocol specifications; otherwise, the processor is said to be faulty [47] . There are two failure types, namely the dormant fault and the malicious fault (also called the Byzantine fault or the arbitrary fault) [33] [50] .

The dormant faults include crashes and omission. A crash fault occurs when a processor stops executing prematurely and does nothing afterward [57] . An omission fault takes place when a processor fails to transmit or receive a message on time or at all. The malicious fault is the most damaging failure type because the behavior of a malicious faulty processor is unpredictable and arbitrary (e.g., suffer benign failures, send bogus values in messages, send messages at the wrong time, send different messages to different processors and work in coordination with other faulty processors to prevent fault-free processors from reaching a common value) [13] [32] . Therefore, the malicious fault is the most damaging failure type and causes the worst problem. If the BA and Consensus problems can be solved under malicious fault conditions, the BA and Consensus problems can be solved under other failure type conditions.

2.2.2 The Symptoms of a Faulty Communication Link

The symptoms of a faulty communication link can also be divided into two types. They are dormant fault (omission and delay) and malicious fault [41] [70] . In a synchronous system, each fault-free processor can detect messages from a dormant faulty communication link using the time-out mechanism or encryption technologies. Messages from a malicious faulty communication link can be detected by encryption technologies.

2.3 Agreement Problems Definition

In this section, we give the definitions of agreement problems. There are Byzantine Agreement (BA) problem, Consensus problem, and Fault Diagnosis Agreement (FDA) problem.

2.3.1 Byzantine Agreement Problem

The Byzantine Agreement (BA) problem was first described and solved by Pease, Shostak, and Lamport [37] . In the classical BA problem, there is a set of generals of the Byzantine army camped with their troops around an enemy city. The generals can only communicate with each other through messengers. To conquer the enemy city, the generals must reach a common agreement on whether or not to launch a united attack at dawn (attack or retreat). It is very important that all the loyal generals should decide on the same agreement, since an attack called by only a small number of the generals would result in a lost battle [28] [37] [47] . The above abstract assumptions can be transferred to corresponding assumptions in the distributed system as follows: In the BA problem, there are n (n≥4) processors in the network, where one processor is designated as the commander that holds an initial value v_s. The commander first sends the initial value v_s to all other processors. On receipt of the value v_s, each processor (without the commander) exchanges the received value with other processors. In addition, there is an adversary that controls up to p_m ($n \geq 3\ p_m + 1$) of the processors and can arbitrarily deviate from the designated protocol specification. After a number of message exchange rounds, a common agreement can be reached among all fault-free processors if and only if the maximum number of faulty processors allowed, p_m, is smaller than one third of the total number of processors in the network ($p_m \leq \lfloor (n-1)/3 \rfloor$). Here, the number of message exchange rounds is $t+1$ ($t \leq \lfloor (n-1)/3 \rfloor$) [28] [37] . In short, to solve a BA problem is to make every fault-free processor be agreed on a common value regardless the influence of faulty components. More precisely, the BA problem is defined by the two properties:

(BA_Agreement): ***All*** fault-free processors agree on a common value;

(BA_Validity): If the source (commander) processor is fault-free, then ***all*** fault-free processors agree on the initial value that the source processor sends.

That is, only one processor has initial value in the BA problem. After executing BA protocol, each fault-free processor can make the same decision (common value), the common value is necessary to be selected from the initial value or the default value (the default value is predefined).

The BA problem has been extensively studied over the past two decades. Many graceful BA protocols have been proposed with various network structure assumptions and different symptoms of faulty components assumptions [28] [37] [51] [55] [60] [61] [63] [66] [70] . They are BA problem in pure wired network and BA problem in pure wireless network.

2.3.1.1 The BA Problem in Pure Wired Network

As indicated in chapter 2.3.1, the BA problem was first described and solved by Pease, Shostak, and Lamport [37] . Subsequently, may graceful BA protocols have been proposed. In [28] [37] , the network topology is fully connected network, and the fallible component assumption involves only malicious faulty processors. In [70] , the network topology is also fully connected network, but the fallible component assumptions are malicious faulty processors and malicious faulty communication links. In [60] , the assumption in a generalized connected network is that all network processors are partitioned into groups. Each group has the same number of processors, with the network topology fully connected. The fallible components include malicious faulty processors and malicious faulty communication links. In the general network [51] , the network topology may not be fully connected, the fallible components are dormant/malicious faulty processors, and dormant/malicious faulty communication links. In the multicasting network [61] , processors are partitioned into groups, with each group having different numbers of processors, the network topology may not be fully connected and fallible components are dormant/malicious faulty processors and dormant/malicious faulty communication links. Table 2-1 shows a comparison of various BA protocols over different wired network models.

Table 2-1 The comparison of various BA protocols over different wired network models

	Network models			
	Fully Connected Network	Generalized Connected Network	General Network	Multicasting Network
Lamport et al.[28]	V			
Pease et al. [37]	V			
Yan et al. [70]	V			
Wang et al. [60]	V	V		
Siu et al. [51]	V		V	
Wang et al. [61]	V	V	V	V

2.3.1.2 The BA Problem in Pure Wireless Network

In recent years, Mobile Ad Hoc Networks (MANETs) have become more popular. MANET is a self-organizing multi-hop system without any fixed infrastructure. Therefore, users can still use network services while continually on the move. Some of the potential MANET applications include military purpose, rescue and conference [39] .

So far, numerous routing algorithm have been proposed for MANET, and they may be generally classified into three categories, table-driven routing algorithm, on-demand routing algorithm and hybrid routing algorithm [22] . The table-driven (also called as the proactive) routing algorithm involves all processors trying to have complete knowledge of all paths to all other processors in the MANET, such like FSR [38] , FSLS [44] , OLSR [24] and TBRPF [6] The on-demand (also called as the reactive) routing algorithm involves paths being discovered when they are required, such like AODV [40] and DSR [25] . The hybrid routing algorithm involves a combination of the table-driven routing algorithm and on-demand routing algorithm, such like ZRP [36] . Any of the above strategies can help each processor transmit a message to other processors in the MANET.

To achieve perfect reliability in the MANET, reaching a common agreement in the presence of faults before performing some special tasks is essential [47] . However, previous BA protocols for static networks [28] [37] [51] [60] [61] [70] are not applicable in an MANET. Because, previous BA problem is considered for a static network [28] [37] [51] [60] [61] [70] , and each processor does not have mobility in a static network. However, each processor has mobility in the MANET, so the processor may move away from or back to the MANET at any time. Therefore, some of the fault-free processors (mobile processors may move away from the network during BA protocol execution and back to the network before ending the BA protocol) would not receive enough messages to reach a common agreement. Thus, previous BA protocols for static networks [28] [37] [51] [60] [61] [70] are not applicable in an MANET.

For MANET, we proposed a BA protocol [67] in 2004, called Mobile Ad-Hoc Agreement Protocol (MAHAP), for solving the BA problem in the MANET. MAHAP is the first BA protocol to solve the BA problem in the MANET. MAHAP allows *return processors* (mobile processors move away from the network during BA protocol execution and back to the network before ending the BA protocol) to reach the same agreement value. The detailed description of MAHAP is shown in chapter 4. Moreover, Wang et al. [63] proposed a BA protocol in 2004, called Byzantine Agreement under Mobile Network, for solving the BA problem in the mobile network. The feature of [47] is that the number of rounds of message exchange required is dynamic. If processor immigrates or moves away from the network, the number of rounds required is recomputed. Subsequently, Wang et al. [66] proposed a BA protocol in 2007, called Subnet Byzantine Agreement Protocol, for solving the BA problem in the virtual subnet of a MANET. The virtual subnet is designed to prevent the broadcast storm problem. However, they assumed that each processor can not leave from the network or immigrate into the network during BA protocol execution. Hence, they did not consider the *return processors*. In Tsou [55] , they proposed another BA protocol in 2007, called Hierarchical Clustering Agreement Protocol, for solving the BA problem in the hierarchical cluster-oriented MANET. Tsou [55] used the cluster concept to make *return processors* receive enough messages to reach a common agreement. Hence, the protocol proposed by Tsou [55] is applicable with *return processor* in cluster-oriented MANET. However, the protocol proposed by Tsou [55] is inapplicable in mobile ip network and MANET when *return processor* is considered. Due to mobile ip network and MANET are not logically divided into cluster architecture. Table 2-2 shows a comparison of various BA protocols over different wireless network models.

Table 2-2 The comparison of various BA protocols over different wireless network models

	Network models				Return processor	
	Mobile IP Network	MANET	Virtual Subnet of MANET	Cluster-oriented MANET	Applicable	Inapplicable
MAHAP	V	V			V	
Wang et al. [63]	V	V			V	
Wang et al. [66]	V	V	V			V
Tsou [55]				V	V	

2.3.2 Consensus Problem

The Consensus problem [3] [52] [64] [65] [70] [71] [72] and BA problem are two closely related fundamental problems in agreement on a common value in distributed system. The difference between the BA problem and Consensus problem is that each processor has its own initial value in the Consensus problem [70] . The Consensus problem is defined by these two properties:

(Consensus_Agreement): ***All*** fault-free processors agree on a common value;

(Consensus_Validity): If the initial value of all processors is v_i, then ***all*** fault-free processors shall agree on v_i.

That is, each processor has its own initial value and the initial value of each processor may be different in the Consensus problem. After executing the Consensus protocol, each fault-free processor can make the same decision (common value), so the common value is necessary to be selected from one of the initial values or the default value (the default value is predefined).

In view of the definition of the initial value and the common value, the Consensus problem is solved if *n* copies of the BA protocol are run in parallel [70] . Through these properties, it can be clearly understood that the BA problem is a special case of the Consensus problem in which only one processor's initial value is of interest. To summaries the differences among the BA problem and Consensus problem, Table 2-3 compares the assumptions and the goals of the BA and Consensus problems.

Table 2-3 The differences among the BA problem and Consensus problem

	Byzantine Agreement	Consensus
The owner of the initial value	Source processor	Each processor
Value to be agreed	A single value	A single value

In previous results [3] [52] [65] [64] [70] [71] [72] , the Consensus problem was also solved in many network models with various fallible component assumptions. They are Consensus problem in pure wired network and Consensus problem in pure wireless network.

2.3.2.1 The Consensus Problem in Pure Wired Network

The Consensus problem has been visited in many kinds of wired networks, such as broadcast network [3] , fully connected network [70] , general network [52] and multicasting network [65] .

For a broadcast network, Babaoglu et al. [3] proposed a protocol to solve the Consensus problem in the presence of malicious faulty processors. Subsequently, Yan et al. [70] proposed a protocol to solve the Consensus problem with malicious faulty processors in fully connected network. Afterward, Siu et al. [52] proposed a protocol for handling dormant/malicious faulty processors and dormant/malicious faulty communication links in general network. For a multicasting network, Wang et al. [65] proposed a protocol to solve Consensus problem with dormant/malicious faulty communication links. Table 2-4 shows a comparison of various Consensus protocols over different network models.

Table 2-4 The comparison of various Consensus protocols over different wired networks

	Network models			
	Broadcast Network	Fully Connected Network	General Network	Multicasting Network
Babaoglu et al. [3]	V			
Yan et al.[70]	V	V		
Siu et al. [52]	V	V	V	
Wang et al. [65]	V	V	V	V

2.3.2.2 The Consensus Problem in Pure Wireless Network

The Consensus problem has also been visited in many kinds of wireless networks, such as mobile ip network [64] , MANET [64] , virtual subnet of MANET [72] and cluster-oriented MANET [71] .

For mobile ip network and MANET, Wang et al. [64] proposed a protocol to solve the Consensus problem with dormant/malicious communication links. For virtual subnet of

MANET, Yan et al. [72] proposed a Consensus protocol with dormant/malicious communiation links. Moreover, Yan et al. [71] proposed a protocol to solve the Consensus problem with dormant/malicious communication links in cluster-oriented MANET. Table 2-5 shows a comparison of various Consensus protocols over different wireless network models.

Table 2-5 The comparison of various Consensus protocols over different wireless networks

	Network models			
	Mobile IP Network	MANET	Virtual Subnet of MANET	Cluster-oriented MANET
Wang et al. [64]	V	V		
Yan et al. [72]			V	
Yan et al. [71]				V

2.3.3 Fault Diagnosis Agreement Problem

In a highly reliable fault-tolerant environment, to reach a common agreement is not enough. It is also necessary to detect/locate the faulty components in the network. Therefore, we must consider another related problem which is called the fault diagnosis problem. There are two fault diagnosis models in the fault diagnosis problem. They are non-agreement [1] [30] [46] and agreement [23] [59] [62] . In the non-agreement fault diagnosis model, one or more fault-free processors can detect the faulty processors, but the detection results of a fault-free processor may not agree with other fault-free processors. In the agreement fault diagnosis model, the detection results of each fault-free processor are the same. Hence, the agreement fault diagnosis model is better than non-agreement fault diagnosis model in a highly reliable fault-tolerant environment. Therefore, we consider the Fault Diagnosis Agreement (FDA) problem [23] [59] [62] . FDA is used to make each fault-free processor detect/locate “the same” faulty components in the network. After reaching the FDA, each fault-free processor can maintain the performance and integrity of the network. Protocols designed to deal with the FDA problem should satisfy the following requirements:

(FDA_Agreement): ***All*** fault-free processors must identify the common set of faulty processors;

(FDA_Fairness): ***NO*** fault-free processor is incorrectly detected as faulty by any fault-free processor.

There are two distinct approaches to dealing with the fault diagnosis problem: the test-based approach [30] and the evidence-based approach [1] [23] [46] [59] [62] . In the test-based approach, where a testing processor *p* may test another processor *q*, if the testing processor *p* is fault-free, then the test results is correct. However, such an approach is not applicable to the case where the testing processor is faulty or the processor *q* is of the malicious kind. The detailed description of malicious fault is shown in chapter 2.2.1. The reason is that if the testing processor *p* is faulty, then the test results of the testing processor *p* will be incorrect; if the processor *q* is of the malicious kind, then the processor *q* can hide its faulty behaviors and pass the test held by the testing processor *p*. Therefore, the test-based approach is not suitable for the fault diagnosis problem with malicious faulty components. On the other hand, in the evidence-based approach, the evidence-based protocol collects the received messages in the BA protocol. Therefore, the evidence-based fault diagnosis protocol uses the received messages in the BA protocol as the evidence to find out the faulty components. Therefore, the evidence-based fault diagnosis protocol can handle the faulty components of the malicious kind [1] [23] [46] [59] [62] . Table 2-6 shows different approaches of the fault diagnosis problem.

After the set of faulty processors is detected/located by all fault-free processors, we can re-configure the network and eliminate the faulty processors to enhance the performance and strengthen the integrity of the network.

Table 2-6 The different approaches of the fault diagnosis problem

	Approaches		Agreement	
	Test-based	Evidence-based	Non-agreement	Agreement
Mallela et al. [30]	V		V	
Adams et al. [1]		V	V	
Shin et al. [46]		V	V	
Hsiao et al.[23]		V		V
Wang et al.[59]		V		V
Wang et al.[62]		V		V

2.3.3.1 The Related Work on the Fault Diagnosis Agreement Problem

In previous result, the FDA problem is only considered for a static network (fully connected network, general network and multicasting network) [1] [23] [30] [46] [59] [62] .

For a fully connected network, Mallela et al. [30] proposed a protocol to detect dormant faulty processors. In their protocol, they used the test-based approach. However, test-based approach is not applicable with malicious faulty processors. Subsequently, Adams et al. [1] and Shin et al.[46] proposed two evidence-based protocols to detect malicious faulty processors in fully connected networks. However, their protocols are non-agreement fault diagnosis model. Hence, the detection results of a fault-free processor may not agree with other fault-free processors. Afterward, Wang et al. [59] proposed a fault diagnosis agreement protocol to detect the malicious faulty processor by evidence-based approach. Hence, the detection results of each fault-free processor are the same.

For a general network, Hsiao et al. [23] proposed a fault diagnosis agreement protocol to detect the malicious faulty processor by evidence-based approach. For a multicasting network, Wang et al. [62] proposed a fault diagnosis agreement protocol to detect the malicious faulty processors by evidence-based approach. Table 2-7 shows a comparison of various FDA protocols over different static network models.

Table 2-7 The comparison of various FDA protocols over different static network models

	Network Models			Agreement	
	Fully Connected Network	General Network	Multicasting Network	Non-agreement	Agreement
Adams et al. [1]	V			V	
Mallela et al. [30]	V			V	
Shin et al. [46]	V			V	
Wang et al.[59]	V				V
Hsiao et al.[23]	V	V			V
Wang et al.[62]	V		V		V

2.4 Conclusion

As described in chapter 2.3, most of existing protocols are designed for solving the BA, Consensus and FDA problems in pure wireless network or pure wired network. However, most network environments today are combined wired and wireless. In this dissertation, we revisited agreement problems in combined wired/wireless network.

Chapter 3

Basic Concepts and Approaches

The research objective of this dissertation is to propose protocols for solving agreement problems in combined wired/wireless network. As a matter of fact, pure wired networks and pure wireless networks are all special cases of the combined wired/wireless networks. We also discuss the agreement problems in pure wired network (fully connected network, broadcasting network, generalize connected network, general network) and pure wireless network (MANET) in chapter 2.3.1, 2.3.2, 2.3.3, and 4. Moreover, we also propose a fault diagnosis agreement protocol to detect/locate the faulty processors. In the usage, many P2P networks are overlay networks because they run on top of the combined wired/wireless network. Hence, we give an application of Consensus protocol to ensure the file consistency of file-sharing in P2P networks. The flow chart of the proposed approaches is shown in Figure 3-1.

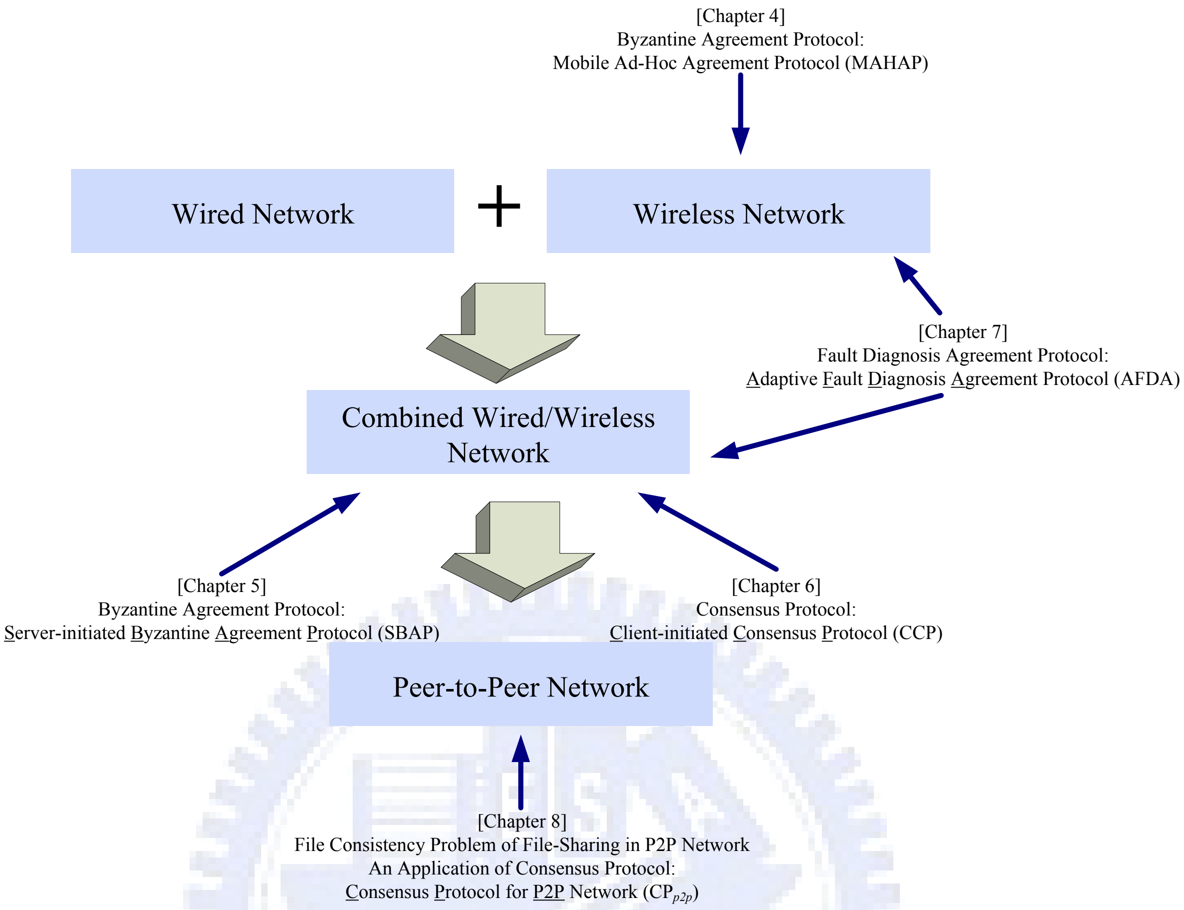

[Chapter 4]
Byzantine Agreement Protocol:
Mobile Ad-Hoc Agreement Protocol (MAHAP)
Wired Network
+
Wireless Network
[Chapter 7]
Fault Diagnosis Agreement Protocol:
Adaptive Fault Diagnosis Agreement Protocol (AFDA)
Combined Wired/Wireless Network
[Chapter 5]
Byzantine Agreement Protocol:
Server-initiated Byzantine Agreement Protocol (SBAP)
[Chapter 6]
Consensus Protocol:
Client-initiated Consensus Protocol (CCP)
Peer-to-Peer Network
[Chapter 8]
File Consistency Problem of File-Sharing in P2P Network
An Application of Consensus Protocol:
Consensus Protocol for P2P Network (CP$_{p2p}$)

Figure 3-1. The flow chart of the proposed approach

3.1 Agreement Problem in Pure Wireless Network

Since, previous BA protocols [28] [37] [51] [60] [61] [70] for wired network are not applicable in an MANET. Hence, we proposed a first BA protocol to solve the BA problem in an MANET in 2004. The first protocol designed to solve the BA problem in an MANET is called MAHAP (Mobile Ad-Hoc Agreement Protocol). MAHAP allows each fault-free processor (include *return processors*) to reach a common agreement value for solving the BA problem using the minimum number of message exchanges and tolerating a maximum number of faulty processors. The assumptions and parameters of the proposed BA protocol in the MANET are listed as follows:

- Each processor in the MANET can be uniquely identified.

- Let N be the set of all processors in the network and $|N| = n$, where n is the number of processors in the underlying MANET, and $n \geq 4$.
- One of processors is designated as the source processor that holds an initial value.
- The processors of the underlying MANET are assumed fallible.
- A processor that transmits messages is called a sender processor.
- There is only one source processor that transmits messages in the first message exchange round in the BA problem.
- Let p_m be the maximum number of malicious faulty processors.
- Let p_a be the maximum number of *away processors*.
- The constraint of the number of malicious faulty processors and *away processors* in the MANET is $n>3p_m+p_a$.
- Each processor can transmit a message to all other processors in the MANET through routing protocols (on-demand routing protocol or table-driven protocol or hybrid routing protocol).
- All messages are encrypted. Intermediate components cannot falsify a message from a sender processor to a receiver processor.
- Each processor can detect a processor that is not in the MANET.
- A processor does not know the faulty status of other processors.

The detailed description of MAHAP is shown in chapter 4.

3.2 Agreement Problem in Combined Wired/Wireless Network

Previous BA and Consensus protocols for wired network [3] [28] [37] [51] [52] [60] [61] [65] [70] were not applicable in combined wired/wireless networks. Because, mobile processors may move away from the network during BA protocol execution and back to the network before ending the BA protocol, these mobile processors would not receive enough messages to reach a common agreement value.

Furthermore, the communication overhead of the BA and Consensus protocols are inherently large because the BA and Consensus protocols require numerous rounds to exchange messages [19] . Previous BA and Consensus protocols [28] [37] [51] [52] [60] [70] designed for flat architectures were not efficient because all messages must propagate globally throughout the network.

To solve the BA and Consensus problems in the combined wired/wireless network, create secure communications between each processor and reduce the communication overhead, we propose a secure communication protocol and hierarchical BA and Consensus protocols.

3.2.1 Byzantine Agreement Problem in Combined Wired/Wireless Network

The proposed BA protocol for combined wired/wireless network is called the SBAP (Server-initiated Byzantine Agreement Protocol). SBAP use the hierarchical model concept to reduce the communication overhead and provide secure communications by cryptographic technologies. The assumptions and parameters of the proposed BA protocol in the combined wired/wireless network are listed as follows:

- The underlying network is a two-level combined wired/wireless network.
- The two levels include a higher network level and a zone level.
- The combined wired/wireless network consists of wired backbones and wireless cells that provide access to mobile processors.
- Processors include agreement-server, mobile processor and stationary processor.
- Agreement-server is a powerful and reliable computer with high bandwidth.
- Mobile processor is a processor with mobility.
- Stationary processor is a processor without mobility.

- Let N be the set of all processors in the network and $|N|= n$, where n is the number of processors in the underlying network.
- Let Z_N be the set of all agreement-servers in the network and $|Z_N|= z_n$, where z_n is the number of agreement-servers in the underlying network and $z_n \geq 4$.
- The underlying network is unreliable: messages may be dropped, reordered, inserted or duplicated by faulty processors.
- Each processor in the network can be identified uniquely.
- Let p_m be the maximum number of malicious faulty processors allowed.
- Let z_m be the maximum number of malicious faulty agreement-servers allowed.
- The constraint of the number of malicious faulty agreement-servers in the network is $z_n > 3z_m$.
- All messages are encrypted. Intermediate components cannot falsify a message from a sender processor to a receiver processor.
- A processor does not know the faulty status of other processors in the underlying network.

The detailed description of SBAP is shown in chapter 5.

3.2.2 Consensus Problem in Combined Wired/Wireless Network

The proposed Consensus protocol for combined wired/wireless network called the CCP (Client-initiated Consensus Protocol). CCP also use the hierarchical model concept to reduce the communication overhead and provide secure communications by SRFC (Secure Relay Fault-tolerance Channel). The assumptions and parameters of the proposed Consensus protocol in the combined wired/wireless network are listed as follows:

- The underlying network is un-fully connected.
- The underlying network is a two-level combined wired/wireless network.

- The two levels include a higher network level and a zone level.
- The combined wired/wireless network consists of wired backbones and wireless cells that provide access to mobile processors.
- Processors include consensus-server, mobile processor and stationary processor.
- Clients include stationary processor and mobile processor.
- Consensus-server is a powerful and reliable computer with high bandwidth.
- Mobile processor is a processor with mobility.
- Stationary processor is a processor without mobility.
- Let N be the set of all processors in the network and $|N|= n$, where n is the number of processors in the underlying network.
- Let Z_N be the set of all consensus-servers in the network and $|Z_N|= z_n$, where z_n is the number of consensus-servers in the underlying network and $z_n \geq 4$.
- The underlying network is unreliable: messages may be dropped, reordered, inserted or duplicated by faulty components.
- Each processor in the network can be uniquely identified.
- Let p_m be the maximum number of malicious faulty processors allowed.
- Let p_d be the maximum number of dormant faulty processors allowed.
- Let z_m be the maximum number of malicious faulty consensus-servers allowed, $z_m \leq \lfloor (z_n-1)/3 \rfloor$.
- The constraint of the number of malicious faulty consensus-servers and dormant faulty consensus-servers in the network is $z_n > \lfloor (z_n-1)/3 \rfloor + 2z_m + z_d$.
- Let z_d be the maximum number of dormant faulty consensus-servers allowed.
- Let c_m be the maximum number of malicious faulty communication links allowed in the higher network level.
- Let c_d be the maximum number of dormant faulty communication links allowed in the higher network level.

- Let c be the connectivity of each consensus-server in the higher network level, where $c>c_m+c_d+z_m+z_d$.
- The connectivity c_p of Zone$_p$ must be larger than the number of malicious faulty components (processors and communication links) plus the number of dormant faulty components in Zone$_p$, where $1\leq p\leq z_n$.
- A processor does not know the faulty status of other components in the underlying network.

The detailed description of SBAP is shown in chapter 5.

3.3 Fault Diagnosis Agreement Problem in Combined Wired/Wireless Network

In recent years, combined wired/wireless networks have become more and more popular. In order to provide a highly reliable computing environment for combined wired/wireless network, we shall propose a new protocol to solve the FDA problem in combined wired/wireless networks.

The proposed protocol called Adaptive Fault Diagnosis Agreement Protocol (AFDA). AFDA is an adaptive FDA protocol. AFDA not only can solve the FDA problem in combined wired/wireless network, but also AFDA can solve the FDA problem in other networks. Because, AFDA is an evidence-based protocol that collects the messages accumulated in a BA protocol as evidence and then detects/locates the common set of faulty processors by examining the collected evidence. For example, AFDA with MAHAP (BA protocol for MANET) can sovle the FDA problem in MANET. AFDA with SBAP (BA protocol for combined wired/wireless network) can solve the FDA problem in combined wired/wireless network. The assumptions and parameters of the proposed AFDA protocol in MANET and combined wired/wireless network is shown in chapter 3.1 and 3.2. The detailed description of

AFDA is shown in chapter 7.

3.4 File Consistency Problem of File-Sharing in Peer-to-Peer Systems

In the usage, the file-sharing application has been the most popular application in Peer-to-Peer (P2P) systems. Many P2P networks are overlay networks because they run on top of the combined wired/wireless network. We know that P2P systems are flexible and self-organizing in adapting to changes in a distributed environment. These adaptive features make the system vulnerable [35] [49] . The malicious processors could work in coordination with other faulty processors to modify the files and spread the inconsistent files to other fault-free processors. If not properly controlled, fault-free processor may also spread inconsistent files to other processors and even potentially paralyze the entire P2P network. Inconsistent file not only spread in P2P networks but also waste resources, such as bandwidth, space of storage, and transmission time.

Hence, checking file consistency with other fault-free processors has become the most challenging problem in a P2P system. In this dissertation, we give an application of Consensus protocol to ensure the file consistency of file-sharing in P2P systems. The proposed protocol is called the CP$_{p2p}$ (Consensus Protocol for P2P Network). CP$_{p2p}$ allows each fault-free processor to ensure the file consistency of file-sharing in P2P systems. The assumptions and parameters of the proposed Consensus protocol in the P2P network are listed as follows:

- Each processor in the P2P network can be uniquely identified.
- Let N be the set of all processors in the network and $|N| = n$, where n is the number of processors in the underlying P2P network, and $n \geq 4$.
- The processors of the underlying P2P network are assumed fallible.
- The maximum number of malicious faulty processors is $n/3$.

- Each fault-free processor has the correct file information.
- A processor does not know the faulty status of other processors.

The detailed description of CP_{p2p} is shown in chapter 8.

Chapter 4
Byzantine Agreement Protocol for Wireless Networks

In this chapter, we introduce our approach to solve the BA problem with malicious faulty processors in a wireless network.

4.1 The Conditions for BA Problem in Wireless Network

To solve the BA problem in wireless network, the system model, the number of required message exchange rounds and the constraint in wireless network should be considered first.

4.1.1 System Model

Because the processors of wireless network (MANET) are mobile, the processors may move away from the network or return at any time. In our system, a processor that moves away from the MANET in the message-exchanging phase is called an "*away processor*", while a processor that returns to the MANET before the decision-making phase is called a "*return processor*". Because an *away processor* moves away from the MANET, an *away processor* cannot transmit and receive messages from other processors in the MANET. The detailed descriptions of the message-exchanging phase and decision-making phase will be presented in chapter 4.2.1 and 4.2.2.

In this chapter, the BA problem is considered in an MANET with fallible processors. The failure type of a fallible processor is malicious (worst case). An MANET example is shown in Figure 4-1. The assumptions and parameters of the system are listed in chapter 3.1.

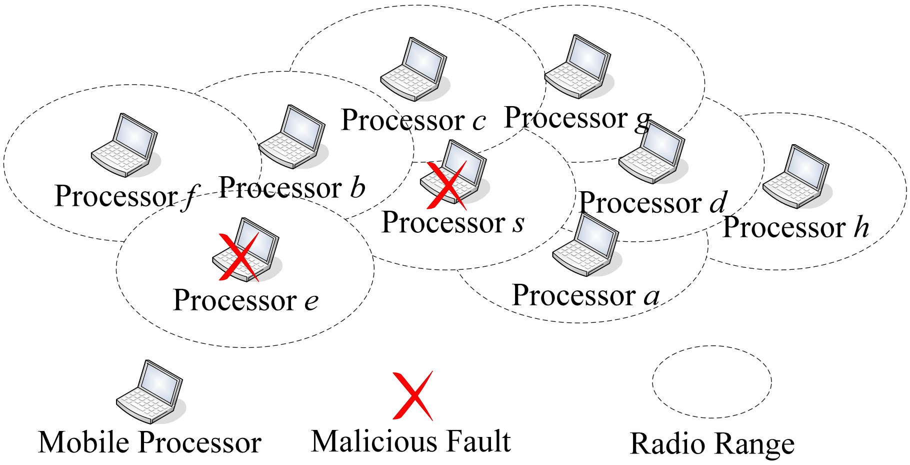


Figure 4-1. An example of MANET

4.1.2 The number of Message Exchange Rounds Required by MAHAP

In the BA problem, each processor exchanges messages during the message-exchanging phase. Thus, the message-exchanging phase is a time consuming phase. Therefore, reducing the number of required rounds is the major concern in an optimal protocol design. The term "round" denotes the message exchange interval between any pair of processors [19] [28] . A round is defined as follows: (1) Sends messages to any set of processors (2) Receives messages from this round (3) Does local processing [19] [28] . Fischer and Lynch [19] indicated that $t+1$ ($t=\lfloor(n-1)/3\rfloor$) is the minimum number of rounds required to get enough messages to reach a common agreement value. By the definition of round, the number of required message exchange rounds in the MANET is also $t+1$ ($t\leq\lfloor(n-1)/3\rfloor$). A detailed description of the message-exchanging phase will be presented in chapter 4.2.1.

4.1.3 Constraint

The number of faulty processors allowed in the network depends on the total number of processors in the network and the processor failure types. For example, in Lamport et al. [28] , the assumption of processor fault type is malicious in a static network. The Lamport et al.

constraint [28] is $n>3p_m$, where p_m is the number of malicious faulty processors.

The BA protocol MAHAP is designed for an MANET with malicious faulty processors. Because MANET processors are mobile and the *away processors* can be detected by each processor in the MANET, the constraint of the system is $n>3p_m+p_a$.

4.2 Proposed BA Protocol: "Mobile Ad-Hoc Agreement Protocol" (MAHAP)

The BA protocol involves making each fault-free processor agree on a common value transmitted by the source processor. Therefore, there are three phases in the MAHAP: message-exchanging phase, decision-making phase and extension-agreement phase. The message-exchanging phase is used to collect the messages, the decision-making phase is used to compute a common agreement value for the BA problem and the extension-agreement phase is used to allow *return processors* to compute a common agreement value that is the same as that of other fault-free processors' agreement value. In addition, the number of rounds required for executing MAHAP is $t+1$ ($t\leq\lfloor(n-1)/3\rfloor$). The MAHAP protocol can tolerate p_m malicious faulty processors, and p_a *away processors*, where $n>3p_m+p_a$. The MAHAP protocol definition is shown in Figure 4-2.

4.2.1 Message-Exchanging Phase

The goal of the message-exchanging phase is to collect the messages. In the MANET, each processor has partial or complete common knowledge of the graphic information of the underlying MANET (this depends on the routing protocols: table-driven or on-demand or hybrid). Each processor can transmit message(s) to other processors in the MANET directly or through intermediate processors (relay processors). To prevent the message falsification by the relay processors, the message is authenticated. Therefore, a message from a sender processor to a receiver processor cannot be falsified by faulty relay processors. If no message is received from a sender processor, the value δ^0 is used as the received message. The value δ^0 is used to report an absent value.

In the message-exchanging phase, the number of rounds required γ must be computed, where γ=t+1, and t=$\lfloor(n\text{-}1)/3\rfloor$. In the first round of the message-exchanging phase, the source processor transmits its initial value v_s using encryption technology to all other processors. Each processor then stores the value from the source processor in the root s of its message gathering tree (mg-tree). The mg-tree is a tree structure that is used to store received messages (a detailed description of the mg-tree is presented in chapter 4.2.4). If the initial value v_s from the source processor is " δ^0", the value "0" is then used to replace the value " δ^0". After the first round of message-exchanging phase (i>1), each processor (except the source processor) transmits the values at level i-1 in its mg-tree using encryption technology to all other processors. If the value at level i-1 is δ^j the value δ^{j+1} is used as the transmitted value, where 0≤j≤$\lfloor(n\text{-}1)/3\rfloor$-1. Each processor stores the values received at level i in its mg-tree.

4.2.2 Decision-Making Phase

The goal of the decision-making phase is to compute a common agreement value for the BA problem. After the message-exchanging phase, each processor has its own mg-tree. Each processor reorganizes its mg-tree into a corresponding information collecting tree (ic-tree). The ic-tree is a tree structure that is used to store received messages without repeated processor names (a detailed description of the ic-tree is presented in chapter 4.2.5). Using the VOTE$_{ad}$ function on each processor's ic-tree from the level t+1 to root s obtains the agreement value VOTE$_{ad}$(s). The agreement value VOTE$_{ad}$(s) is transmitted to the *return processors*. The formal description of the VOTE$_{ad}$ function is shown in Figure 4-3. There are five conditions in the VOTE$_{ad}$ function. If the vertex α is a leaf, then there is only one value in the vertex α. Thus, the majority value is the value of vertex α (condition 1). Condition 2 is used to remove the influence from malicious faulty processors. Condition 3 is used to remove the influence from no response processors and presents the existence of absentees. Condition 4 is used to get the majority value. Condition 5 happens when there is no majority value. Conditions 1, 4, and 5 in the VOTE$_{ad}$ function are similar to the conventional majority vote [28] .

4.2.3 Extension-Agreement Phase

The goal of the extension-agreement phase is to allow *return processors* to compute a common agreement value the same as that of other fault-free processors' function VOTE$_{ad}$(s) value. In the extension-agreement phase, each *return processor* receives the agreement values from the other processors. The VOTE$_{ad}$ function is used on the values received to obtain the agreement value. The *return processors* can then obtain the same agreement value as other fault-free processors. The reason is that each fault-free processor can reach a common agreement value if $n>3p_m+p_a$. Thus, there are at least $n-\lfloor(n-1-p_a)/3\rfloor-p_a$ processors that are fault-free and have the same agreement value. That is, in the worst case, a *return processor* can receive $n-\lfloor(n-1-p_a)/3\rfloor-p_a$ copies of the same value, which is larger than $\lfloor(n-1-p_a)/3\rfloor$. A *return processor* can then decide which agreement value using the VOTE$_{ad}$ function.

MOBILE AD-HOC AGREEMENT PROTOCOL (MAHAP)

Definition:

- For the “Table-driven” Ad-Hoc routing protocols, each processor has common knowledge of the entire graphic information G=(E,N), where N is the set of processors in the network and E is a set of processor pairs (N_i,N_j) indicating a physical link (the radio range is covered) between processor N_i and processor N_j ,where $1 \leq i,j \leq n$.
- For the “On-demand” Ad-Hoc routing protocols, each processor has partial common knowledge of the graphic information G=(E,N).
- Each processor can transmit a message to all other processors in the MANET through the “On-demand“ or the “Table-driven” or the “Hybrid” protocol.
- Three processor roles: sender processor, relay processor and receiver processor, depending on the message flow. A message sent from a sender processor to a receiver processor may be passed through some intermediate (relay processor).
- A relay processor cannot falsify a message from a sender processor to a receiver processor. This is achieved using encryption technology.
- The *return processor*, a processor that moves away from the MANET in the message-exchanging phase, and returns to the MANET before the decision-making phase.
- Each processor receives other processors’ messages at each message exchange round. If no message is received, the value is replaced with δ^0.

Figure 4-2. The proposed MAHAP protocol (cont’d.)

```
Message-Exchanging Phase:
        Compute the number of rounds required $\gamma$: $\gamma=t+1$, where $t=\lfloor(n-1)/3\rfloor$.
    For $i$ = 1 to $\gamma$
      If $i$=1, then:
        1. The source processor transmits its initial value $v_s$ using encryption technology to all other processors.
        2. Each processor stores the value from the source processor in the root $s$ of its mg-tree.
        3. If the initial value $v_s$ from the source processor is "$\delta^0$", the value "0" is used to replace the "$\delta^0$" value.
      If $i > 1$, do:
        1. Except for the source processor, each processor transmits the values at level $i$-1 in its mg-tree using encryption technology to all other processors. If the value at level $i$-1 is $\delta^j$, the value $\delta^{j+1}$ is used as the transmitted value, where $0\leq j\leq\lfloor(n-1)/3\rfloor-1$.
        2. Each processor stores the values received at level $i$ of its mg-tree.
    Next $i$

Decision-Making Phase:
  Step 1: Each processor turns the mg-tree into its corresponding ic-tree by deleting the vertices with repeated names.
  Step 2: Each processor uses the $\text{VOTE}_{ad}$ function on its ic-tree from the level $t$+1 to root $s$ and obtains the agreement value $\text{VOTE}_{ad}(s)$.
  Step 3: Each processor transmits the $\text{VOTE}_{ad}(s)$ value by using encryption technology to the return processors if there is any return processor in the MANET.

Extension-Agreement Phase: (for the return processor only)
  Step 1: Each return processor requests other processors to send their $\text{VOTE}_{ad}(s)$ values.
  Step 2: Each return processor receives other processors' $\text{VOTE}_{ad}(s)$ values.
  Step 3: Each return processor uses the $\text{VOTE}_{ad}$ function to the received messages to obtain the agreement value.
```

Figure 4-2. The proposed MAHAP protocol

```
FUNCTION VOTE_ad(α)
    begin
        if the $\alpha$ is a leaf
            output the value of $\alpha$                          /* condition 1*/
        else begin
            if the number of value $\delta^0$ is ≥ $3*(t-\gamma+1)+[(n-1) \bmod 3]$
                output the value of $\alpha$                      /* condition 2*/
            if the majority value is $\delta^i$, where $1\le i \le \lfloor (n-1)/3 \rfloor -1$
                output the value $\delta^{i-1}$                   /* condition 3*/
            if the majority value is the non- $\delta^j$ value, where $0\le j \le\lfloor (n-1)/3 \rfloor -1$, $m\in\{0,1\}$
                output the majority value $m$                     /* condition 4*/
            if the majority value does not exist
                output the default value $\phi$                   /* condition 5*/
        end
    end.
```

The VOTE$_{ad}$ function only counts the non-value δ^0 (excluding the last level of the ic-tree) for all vertexes at the γ-th level of an ic-tree, where $1\le \gamma \le t+1$, $t=\lfloor (n-1)/3 \rfloor$.

Figure 4-3. The VOTE$_{ad}$ function

4.2.4 The Message Gathering Tree (mg-tree)

The structure of an mg-tree with one level, an mg-tree with two levels and mg-tree with three levels are shown in Figure 4-4(b), Figure 4-4(d) and Figure 4-4(e). Each fault-free processor maintains such an mg-tree during the execution of MAHAP. At the first message exchange round, Processor *s* transmits its initial value to the other processors. We assume that each receiver processor can always identify the sender of a message. When a fault-free processor receives the message sent from the source processor, it stores the received value, denoted as *val*(*s*), at the root of its mg-tree as shown in Figure 4-4(b). At the second message exchange round, each processor transmits the root value of its mg-tree to the other processors. If Processor *a* sends message *val*(*s*) to Processor *b*, then Processor *b* stores the received messages from Processor *a*, denoted as *val*(*sa*), in vertex *sa* of its mg-tree. Similarly, if Processor *b* sends message *val*(*sa*) to Processor *a*, then the value is *val*(*sab*) and stored in vertex *sab* of Processor *a*'s mg-tree as presented in Figure 4-4(e). Generally speaking, message *val*(*sa*…*n*), stored in the vertex *sa*…*n* of an mg-tree, implies that the message just received was sent through the source processor, Processor *a*,…, Processor *n*, where Processor *n* is the latest processor to pass the message. When a message is transmitted through a processor more than once, the name of the processor will be repeated correspondingly. For instance, the appearance of message *val*(*saa*) in vertex *saa* in Figure 4-4(e) indicates that the message is sent from Processor *s* to Processor *a* and to somewhere else and then to Processor *a* again; therefore, Processor *a* appears twice in vertex name *saa*.

In summary, the root of an mg-tree is always named *s* to denote that the stored message is sent from the source processor at the first message exchange round, and the vertex of an mg-tree is labeled with a list of processor names. The processor name list contains the names of the processors through which the stored message was transferred.

4.2.5 The Information Collecting Tree (ic-tree)

An ic-tree is reorganized from a corresponding mg-tree by removing the vertices with repeated processor names in order to avoid the repeated influences from faulty processors in an ic-tree. In Figure 4-4(f), there is an example of an ic-tree created by deleting the repeated processors name of the original mg-tree.

4.3 An MAHAP Execution Example

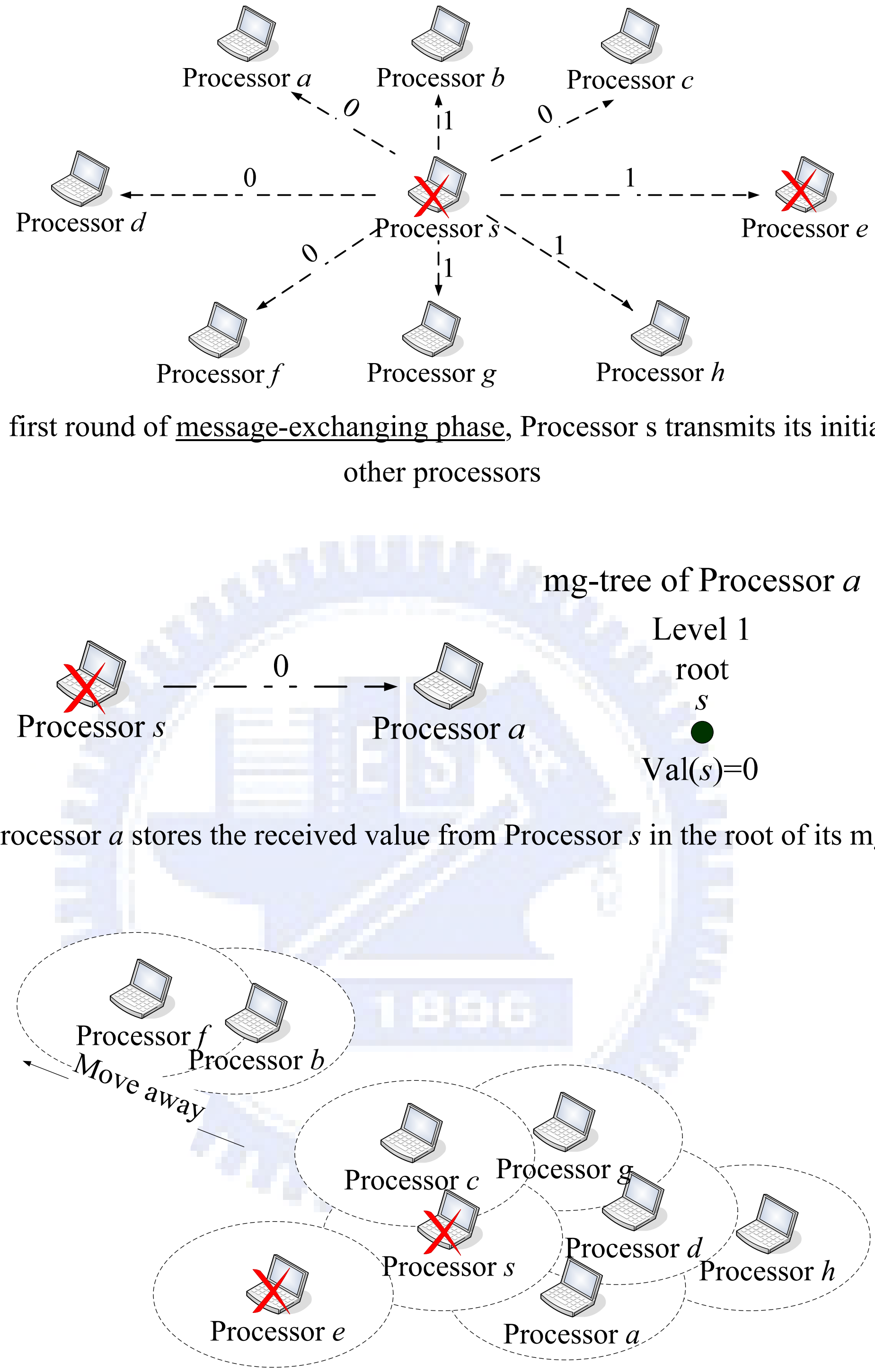
The worst BA problem case occurs when the source processor is a malicious faulty processor. If the BA problem can be solved in the worst case (the source processor is a malicious faulty processor), the BA problem can be solved in all other cases if $n>3p_m+p_a$. Therefore, in this section, an example of executing MAHAP in the worst case is given.

The MANET example is shown in Figure 4-1. There are nine processors in the network. The malicious faulty processors are Processors *s* and *e*. The source processor is a malicious faulty processor which means that Processor *s* may transmit different values to different processors. To reach a common agreement value from each fault-free processor in our example, the MAHAP needs 3 ($\lfloor(9-1)/3\rfloor+1$) message exchange rounds.

In the first round of message-exchanging phase, the source processor Processor *s* transmits different messages to different processors in the MANET, as shown in Figure 4-4(a). Therefore, the fault-free Processor *a* receives the value “0” from the source processor in the first round of message-exchanging phase and stores the message received in the root *s* of its mg-tree, as shown in Figure 4-4(b). In the second round of message-exchanging phase, Processors *b* and *f* move away from the MANET, as shown in Figure 4-4(c). Fault-free Processor *a* therefore cannot receive the message from Processors *b* and *f*. An mg-tree example of Processor *a* in the second round of message-exchanging phase is shown in Figure 4-4(d). In addition, an mg-tree example of Processor *a* in the third round is shown in Figure 4-4(e).

In the decision-making phase, each fault-free processor turns its mg-tree into the corresponding ic-tree by deleting the vertices with repeated names to avoid repeated influence from faulty processors. An example of an ic-tree by Processor *a* is shown in Figure 4-4(f). Using the VOTE_{ad} function on its ic-tree from the level t+1 to the root *s*, an agreement value “0” can be obtained. An example using the VOTE_{ad} function on Processor *a*’s ic-tree from the level t+1 of the ic-tree to roots *s* is shown in Figure 4-4(g). If there is any *return processor* in the MANET, each processor transmits its agreement value to the *return processor*.

In the extension-agreement phase, each *return processor* receives the other processors’ agreement values (the function $\text{VOTE}_{ad}(s)$ values). Using the VOTE_{ad} function on the received messages, an agreement value can be obtained. Processor *b* returns to the MANET before the decision-making phase, as shown in Figure 4-4(h). An example of Processor *b* getting the other processors’ agreement values and using the VOTE_{ad} function to obtain the final agreement value “0” is shown in Figure 4-4(i).

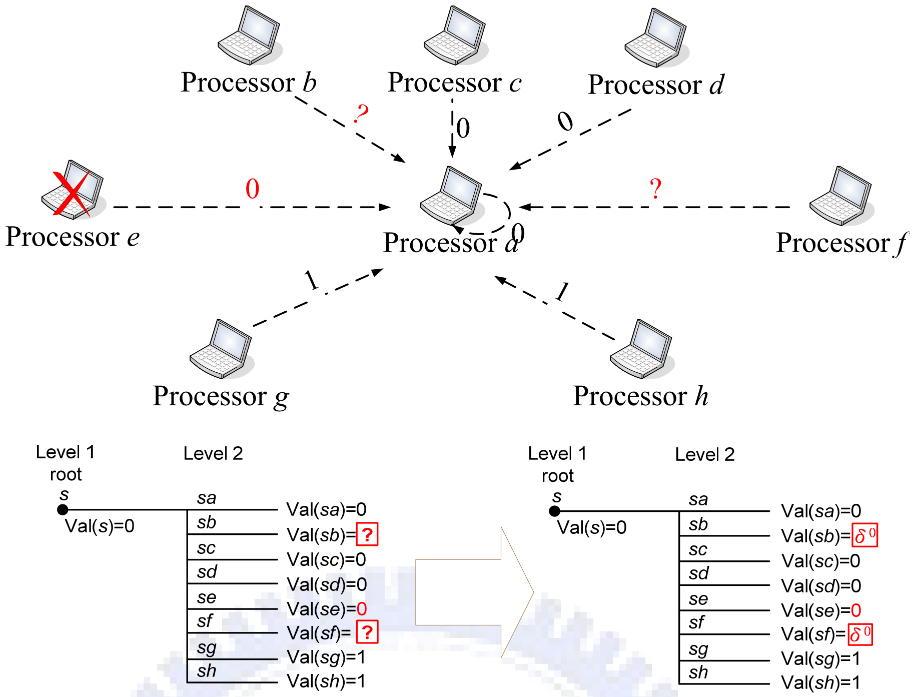


(a) In the first round of message-exchanging phase, Processor s transmits its initial value to other processors

(b) Processor *a* stores the received value from Processor *s* in the root of its mg-tree

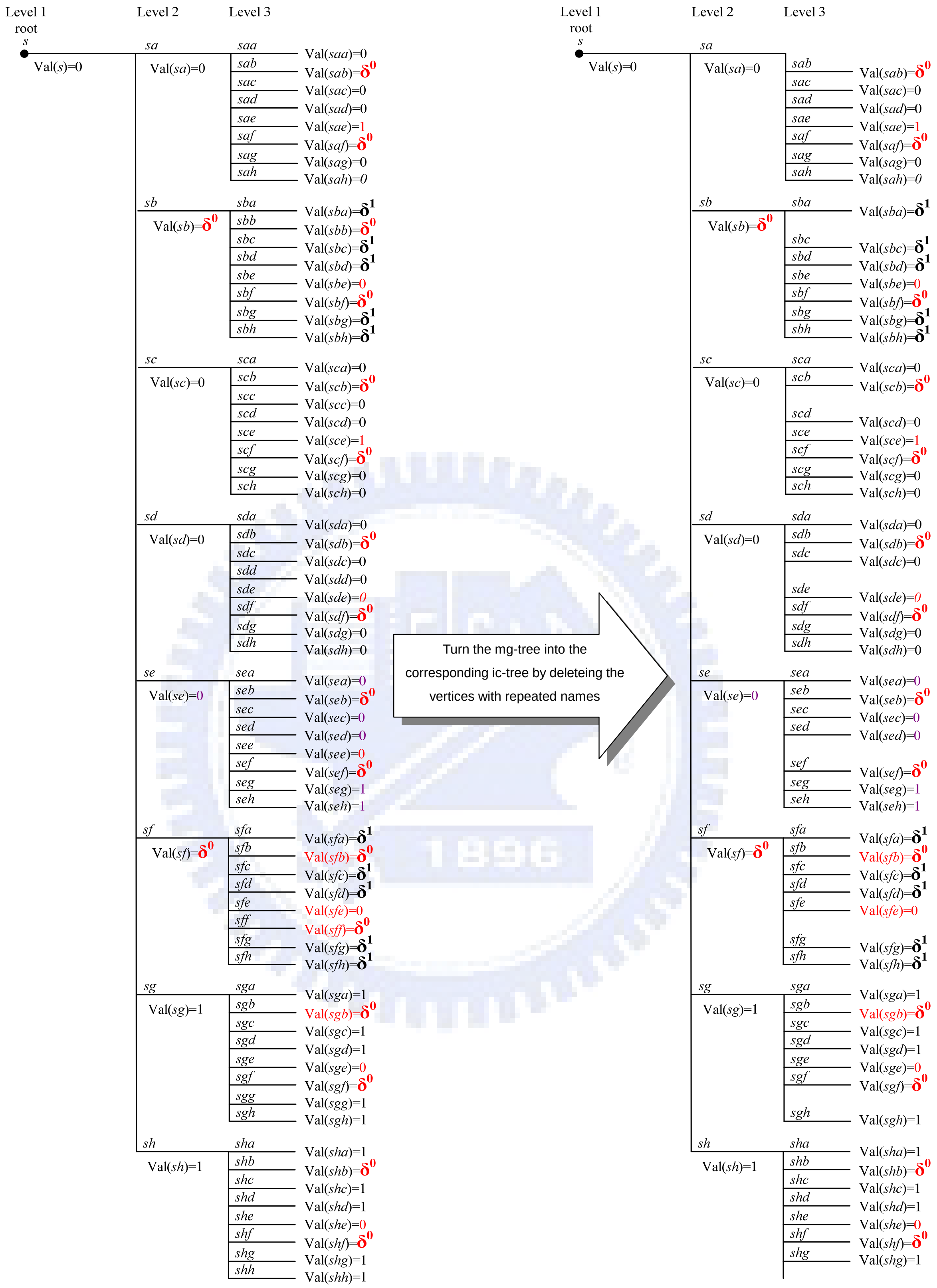
(c) Processor b and f move away in the second round of message-exchanging phase

Figure 4-4. An example of executing MAHAP (cont'd.)



(d) Processor *a* can detect the Processor *b* and Processor *f* do not send the message and stores the received messages from other processors in the level 2 of its mg-tree

Figure 4-4. An example of executing MAHAP (cont'd.)



(e) The final mg-tree of Processor *a* after the message-exchanging phase (mg-tree with three levels)

(f) The ic-tree of Processor *a*

Figure 4-4. An example of executing MAHAP (cont'd.)

◆ **(VOTE(*sa*)= (VOTE(*sab*), VOTE(*sac*), VOTE(*sad*), VOTE(*sae*), VOTE(*saf*), VOTE(*sag*), VOTE(*sah*))**

(VOTE(*sa*) =δ^{0},0,0,1,δ^{0},0,0) = 0 by the condition 4 in the function VOTE

◆ **(VOTE(*sb*)= (VOTE(*sba*), VOTE(*sbc*), VOTE(*sbd*), VOTE(*sbe*), VOTE(*sbf*), VOTE(*sbg*), VOTE(*sbh*))**

(VOTE(*sb*) =δ^{1},δ^{1},δ^{1},0,δ^{0},δ^{1},δ^{1}) = δ^{0} by the condition 3 in the function VOTE

◆ **(VOTE(*sc*)= (VOTE(*sca*), VOTE(*scb*), VOTE(*scd*), VOTE(*sce*), VOTE(*scf*), VOTE(*scg*), VOTE(*sch*))**

(VOTE(*sc*) = 0,δ^{0}, 0,1,δ^{0},0,0) = 0 by the condition 4 in the function VOTE

◆ **(VOTE(*sd*)= (VOTE(*sda*), VOTE(*sdb*), VOTE(*sdc*), VOTE(*sde*), VOTE(*sdf*), VOTE(*sdg*), VOTE(*sdh*))**

(VOTE(*sd*) = 0,δ^{0},0,0,δ^{0},0,0) = 0 by the condition 4 in the function VOTE

◆ **(VOTE(*se*)= (VOTE(*sea*), VOTE(*seb*), VOTE(*sec*), VOTE(*sed*), VOTE(*sef*), VOTE(*seg*), VOTE(*seh*))**

(VOTE(*se*) = 0,δ^{0},0,0,δ^{0},1,1) = 0 by the condition 4 in the function VOTE

◆ **(VOTE(*sf*)= (VOTE(*sfa*), VOTE(*sfb*), VOTE(*sfc*), VOTE(*sfd*), VOTE(*sfe*), VOTE(*sfg*), VOTE(*sfh*))**

(VOTE(*sf*) =δ^{1},δ^{0},δ^{1},δ^{1},0,δ^{1},δ^{1}) =δ^{0} by the condition 3 in the function VOTE

◆ **(VOTE(*sg*)= (VOTE(*sga*), VOTE(*sgb*), VOTE(*sgc*), VOTE(*sgd*), VOTE(*sge*), VOTE(*sgf*), VOTE(*sgh*))**

(VOTE(*sg*) =1,δ^{0},1,1,0,δ^{0},1) = 1 by the condition 4 in the function VOTE

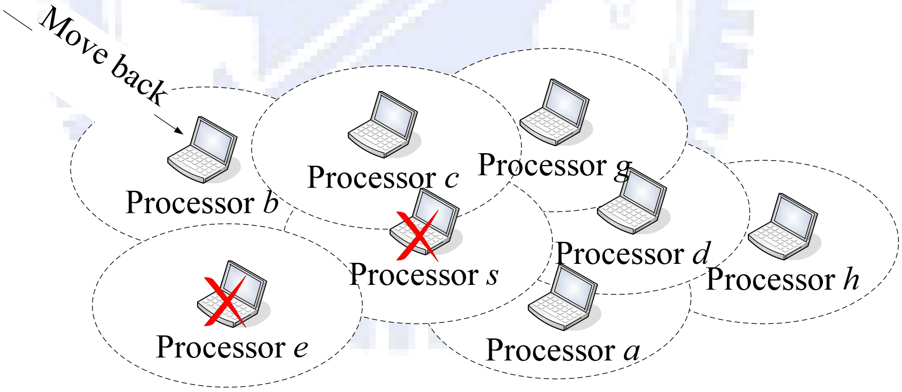
◆ **(VOTE(*sh*)= (VOTE(*sha*), VOTE(*shb*), VOTE(*shc*), VOTE(*shd*), VOTE(*she*), VOTE(*shf*), VOTE(*shg*))**

(VOTE(*sh*) = 1,δ^{0},1,1,0,δ^{0},1) = 1 by the condition 4 in the function VOTE

■ **(VOTE(*s*)= (VOTE(*sa*), VOTE(*sb*), VOTE(*sc*), VOTE(*sd*), VOTE(*se*), VOTE(*sf*), VOTE(*sg*), VOTE(*sh*))**

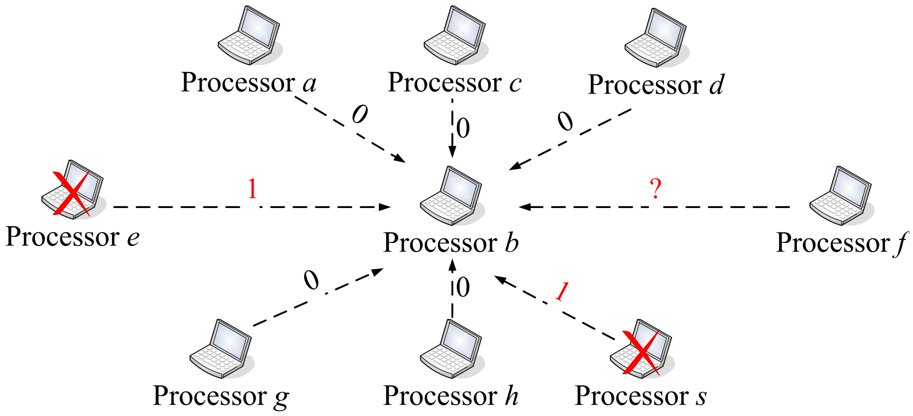
(VOTE(*s*) = 0,δ^{0},0,0,0,δ^{0},1,1) = 0 by the condition 4 in the function VOTE

(g) Using function VOTE$_{ad}$ on Processor *a*'s ic-tree from the level *t*+1 to root *s*, an agreement value "0" can be obtained.



(h) Processor *b* returns to the MANET before the decision-making phase.

Figure 4-4. An example of executing MAHAP (cont'd.)



$\text{VOTE}_{ad}(0,0,0,1, \delta\ 0,0,0,1) = 0$

(i) Processor *b* can get the other processors' agreement values and use function VOTE_{ad} to obtain the final agreement value "0".

Figure 4-4. An example of executing MAHAP

4.4 The Correctness and Complexity of MAHAP

To prove the correctness of our protocol, a vertex α is called common [4] if each fault-free processor has the same value for α. That is, if vertex α is common, then the value stored in vertex α of each fault-free processor's mg-tree or ic-tree is identical. When each fault-free processor has the common initial value from the source processor in the root of its ic-tree, if the root *s* of the ic-tree of a fault-free processor is common and the initial value received from the source processor is stored in the root of the tree structure, then an agreement is reached because the root is common. Thus, the constraints, (BA_Agreement) and (BA_Validity), can be rewritten as:

(BA_Agreement'): Root *s* is common, and

(BA_Validity'): $\text{VOTE}_{ad}(s) = v_s$ for each fault-free processor, if the source processor is fault-free.

To prove that a vertex is common, the term common frontier [4] is defined as follows: When every root-to-leaf path of the tree (an mg-tree or an ic-tree) contains a common vertex, the collection of the common vertices forms a common frontier. In other words, every fault-free processor has the same messages collected in the common frontier if a common frontier does exist in a fault-free processor's tree structure (mg-tree or ic-tree). Subsequently, using the same majority voting function to compute the root value of the tree structure, every fault-free processor can compute the same root value because the same input (the same collected messages in the common frontier) and the same computing function will produce the same output (the root value).

Since MAHAP can solve the BA problem, the correctness of MAHAP should be examined using the following two terms.

(1) Correct vertex: Vertex αi of a tree is qualified as a correct vertex if processor *i* (the last processor name in vertex αi's processor name list) is fault-free. In other words, a correct vertex is a place to store the value received from a fault-free processor.

(2) True value: For a correct vertex αi in the tree of a fault-free processor *j*, val(αi) is the true value of vertex αi. In other words, the stored value is the true value.

According to the definition of a correct vertex, the value it stores is received from a fault-free processor, and a fault-free processor always transmits the same value to all processors. Therefore, the correct vertices of such an mg-tree are common. After turning the mg-tree into its corresponding ic-tree by deleting the vertices with repeated processor names, the values stored on the correct vertices of an ic-tree will be the same. Therefore, all of the correct vertices of an ic-tree are also common. Again, using the definition of a correct vertex, a common frontier does exist in the ic-tree. Thus, the root can be proven a common vertex [(BA_Agreement') is true] due to the existence of a common frontier, regardless of the correctness of the source processor. An agreement on the root value can now be reached. To check the validity of (BA_Validity'), (BA_Validity') is always true due to the propositional logic [8] . If the source processor fails, (BA_Validity') is true, and the reason is that the proposition [(P➔Q)] means (NOT(P) OR Q), so (NOT(P) OR Q) or (P➔Q) is true when P is false, where P implies “the source processor is fault-free” and (P➔Q) implies (BA_Validity’s). Conversely, root *s* is a correct vertex by the definition of a correct vertex if the source processor is fault-free. If all of the correct vertices’ true values can be computed by MAHAP, then the true value of the root can also be computed because the root is a correct vertex. As defined earlier, the true value of the root is the initial value of the source processor if the source processor is fault-free. In short, each fault-free processor’s root value is the initial value of the source processor if the source processor is fault-free; therefore, (BA_Validity’) is true if the source processor is fault-free. Since (BA_Agreement’) and (BA_Validity’) are both true regardless if the source processor is fault-free or failed, the BA problem is solved.

Lemma 4-1: All correct vertices of an ic-tree are common.

Proof: After reorganization, no repeated vertices are in an ic-tree. At level t +1 or above, the correct vertex α has at least $2t$ +1 children, out of which at least t +1 children are correct. The true values of these t +1 correct vertices are common, and the majority of the vertex values α are common. The correct vertex α is common in the ic-tree if the level of α is less then t +1. Consequently, all correct vertices of the ic-tree are common. ■

Lemma 4-2: The common frontier exists in the ic-tree.

Proof: There are t +1 vertices along each root-to-leaf path of an ic-tree in which the root is labeled by the source name, and the others are labeled by a sequence of processor names. Since at most t processors can fail, there is at least one correct vertex along each root-to-leaf path of the ic-tree. Using Lemma 4-1, the correct vertex is common and the common frontier exists in each fault-free processor’s ic-tree. ■

Lemma 4-3: Let α be a vertex, α is common if there is a common frontier in the sub-tree rooted at α.

Proof: If the height of α is 0 and the common frontier (α itself) exists, then α is common. If the height of α is r, the children of α are all in common under the induction hypothesis with the height of the children being r-1. ■

Corollary 4-1: The root is common if the common frontier exists in the ic-tree.

Theorem 4-1: The root of a fault-free processor's ic-tree is common.

Proof: Using Lemma 4-1, Lemma 4-2, Lemma 4-3 and Corollary 4-1, the theorem is proved. ■

Theorem 4-2: Protocol MAHAP solves the BA problem in an MANET.

Proof: To prove this theorem, MAHAP must meet the constraints (BA_Agreement') and (BA_Validity')

(BA_Agreement'): Root s is common.

By Theorem 4-1, (BA_Agreement') is satisfied.

(BA_Validity'): $\text{VOTE}_{ad}(s) = v$ for all fault-free processors, if the initial value of the source is v_t say $v = v_t$

Most processors are fault-free. The value of the correct vertices for all of the fault-free processor mg-trees is v.

When the mg-tree is turned into an ic-tree, the correct vertices still exist. Therefore, each correct vertex of the ic-tree is common (Lemma 4-1), and its true value is v. Using Theorem 4-1, this root is common. The computed value $\text{VOTE}_{ad}(s) = v$ is stored in the root for all the fault-free processors. Therefore, (BA_Validity') is satisfied. ■

Theorem 4-3: MAHAP requires t+1 rounds in the message-exchanging phase to solve the BA problem in a MANET, and t+1 (t=$\lfloor(n$-1)/3)$\rfloor$) is the minimum number of rounds in the message-exchanging phase.

Proof: In the BA problem, each processor exchanges messages during the message-exchanging phase. Thus, the message-exchanging phase is a time consuming phase. Therefore, reducing the number of required rounds is the major concern in an optimal protocol design. The term "round" denotes the message exchange interval between any pair of processors [19] [28] . A round is defined as follows: (1) Sends messages to any set of processors (2) Receives messages from this round (3) Does local processing [19] [28] . Fischer and Lynch [19] indicated that t+1 (t=$\lfloor(n$-1)/3$\rfloor$) is the minimum number of rounds required to get enough messages to reach a common agreement value. By the definition of round, the number of required message exchange rounds in the MANET is also t+1 ($t\leq\lfloor(n$-1)/3$\rfloor$). ■

4.5 Conclusion

In order to provide a reliable computing environment for dynamic MANET, we need to solve the BA problem in the dynamic MANET. To the best of our knowledge, MAHAP is the first protocol to solve the BA problem in the dynamic MANET. In this study, we revisited the BA problem in an MANET with the most damaging type of failed processors (malicious fault). The proposed MAHAP is the optimal protocol that can solve the BA problem in the MANET. The term "optimal" means that the protocol can reach an agreement with the minimum number of rounds required and tolerates the maximum number of faulty components.

MAHAP has the following features:

◆ MAHAP can solve the BA problem in various MANETs (Such as table-driven routing MANET, on-demand routing MANET and hybrid routing MANET) by Theorem 4-2.

- MAHAP allows *return processors* to reach the same agreement value by extension-agreement phase in MAHAP
- MAHAP can solve the BA problem using the minimum number of message exchange rounds (*t*+1 rounds of message exchange)
- MAHAP can tolerate the most damaging failed processor type (malicious fault).
- MAHAP can tolerate p_m malicious faulty processors and p_a *away processors*.

Chapter 5

Server-initiated Agreement Protocol for Combined Wired/Wireless Networks

Wireless networks have become ubiquitous, making combined wired/wireless network a popular trend of development in nowadays. In practice, most current networks are combined wired and wireless environments. In this chapter, we introduce our approach to solve the BA problem with malicious faulty processors in a combined wired/wireless network.

5.1 The Conditions for BA Problem in Combined Wired/Wireless Network

To design a BA protocol in combined wired/wireless network, certain conditions must be taken into account. They are the system model, properties of the BA problem and the constraint in combined wired/wireless network.

5.1.1 System Model

Byzantine agreement protocols imply large communication overhead [19] [28] . The previous network architectures from these results [28] [37] [51] [60] [70] were all flat architectures, with all processors carrying the same responsibility. BA protocols in flat architectures are not efficient because all messages must propagate globally throughout the network. To reduce the communication overhead, we used a hierarchical model concept. Our network model is a two-level combined wired/wireless computing environment consisting of a wired backbone and wireless cells that provide access to mobile processors.

In this chapter, the BA problem is considered in a combined wired/wireless network with fallible processor. The failure type of a fallible component is malicious fault (worst case). Figure 5-1 shows an example of the two-level combined wired/wireless network. There are sixteen processors in the network. There are four agreement-servers, five stationary processors and seven mobile processors. Each agreement-server manages a zone's processors. For example, agreement-server AS_A manages processor A_1, A_2 and A_3 in the zone A. The assumptions and parameters of the system are listed in chapter 3.2.1.

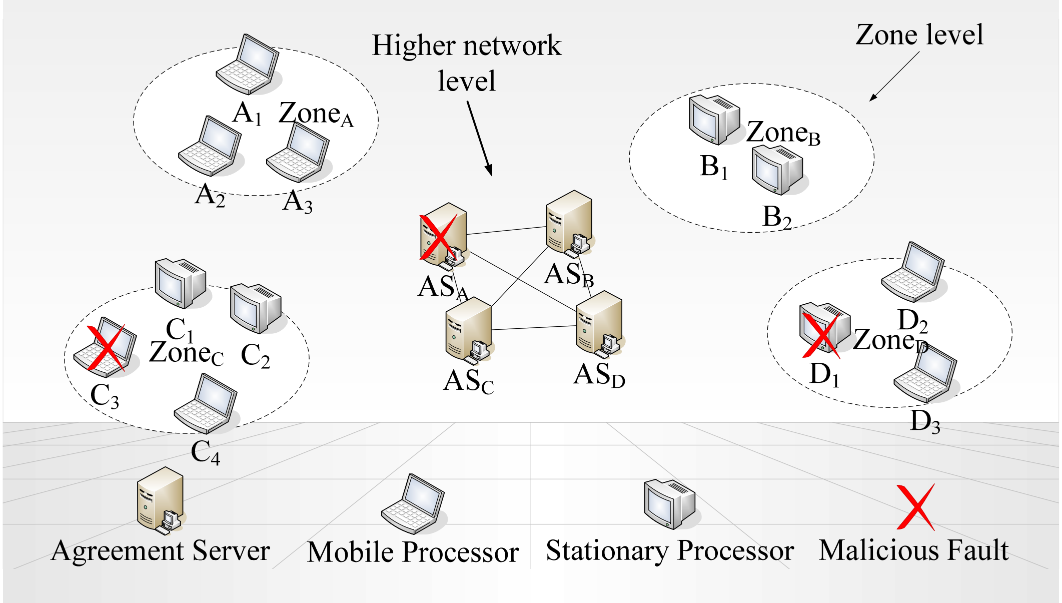


Figure 5-1. An example of combined wired/wireless network

5.1.2 Properties of the BA Problem

By the concept of multilevel hierarchy, each processor in the zone is managed by its agreement-server. Hence, two properties of BA problem in two-level combined wired/wireless network are modified as follows:

(BA_Agreement): All fault-free processors managed by fault-free agreement-server agree on a common value;

(BA_Validity): If the source agreement-servcer is fault-free, the agreement value should be the initial value of the source agreement-server.

5.1.3 Constraint

In the BA problem, the number of faulty processors that can be allowed is determined by the total number of processors in the network. Pease, Shostak and Lamport [28] indicated the constraint of the BA problem is $n>3p_m$.

The network architecture of Pease, Shostak and Lamport [28] is flat architecture; so all processors need to exchange the messages in the message-exchanging phase. In our protocol, the network architecture is hierarchical, only agreement-servers need to exchange the messages in the message-exchanging phase, so the constraint of our model is $z_n>3z_m$.

5.2 Secure Communication

A close study of cryptographic technologies is not necessary for our purpose. Hence, we give a brief introduction of some cryptographic technologies that are used in our system.

5.2.1 Related Cryptographic Technologies

The brief introduction of Diffie-Hellman key exchange, advanced encryption standard and threshold signature are shown here.

5.2.1.1 Diffie-Hellman Key Exchange

Diffie-Hellman key exchange [16] is a cryptographic protocol that allows two processors to agree on a secret key over an insecure communication channel. Once the shared secret key has been established, they can use it to encrypt their secret communication using conventional cryptographic methods. Figure 5-2 shows the Diffie-Hellman key exchange procedure.

5.2.1.2 Advanced Encryption Standard – Symmetric Cryptographic Algorithm

In a symmetric cryptographic system, the communication parties share a key in advance. They encrypt and decrypt delivered messages by the shared key. The security is based on the shared key. If adversaries reveal the shared key, the symmetric cryptographic system will crash. Advanced Encryption Standard (AES) also known as Rijndael [14] is a block cipher adopted as an encryption standard by the US government and expected to be used worldwide. It has been extensively analyzed and compared with its predecessor, the Data Encryption Standard (DES) [45] .

5.2.1.3 Threshold Signature

In threshold signature scheme, the secret *s* is divided into *k* shares and is set a threshold value *h* ($h \le k$). When collecting to *h* above shares, we can reconstruct the original secret *s*. The threshold signature is based on the following equation:

$f(x) = a_{h-1}x^{h-1} + a_{h-2}x^{h-2} + ... + a_1 x + a_0 \bmod p$, where p is a prime number.

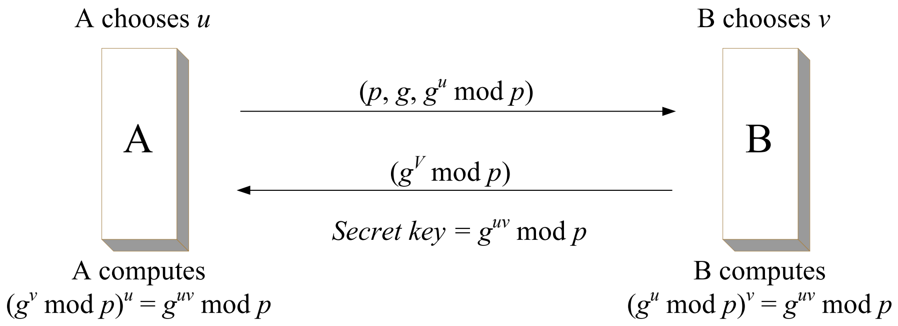

A chooses u
B chooses v
A
B
$(p, g, g^u \bmod p)$
$(g^V \bmod p)$
Secret key $= g^{uv} \bmod p$
A computes
$(g^v \bmod p)^u = g^{uv} \bmod p$
B computes
$(g^u \bmod p)^v = g^{uv} \bmod p$

Figure 5-2. Diffie-Hellman key exchange

5.2.2 Approach

We know that mobile processor power is supplied using batteries. Because power saving is a serious topic with mobile processors. The asymmetric cryptographic algorithm, which needs a large amount of computation is not suited for mobile processors. The advantage of the symmetric cryptographic algorithm is that it is generally much faster than the asymmetric cryptographic algorithm. However, the disadvantage of the symmetric cryptographic algorithm is the requirement for a shared secret key with one copy at each end. Hence, maintaining secure during distribution is an important problem.

Hence, we combined the asymmetric cryptographic and symmetric cryptographic algorithms to obtain the advantages of both in this study. That is, the session key is generated using the Diffie-Hellman key exchange and the symmetric key is generated using the AES. Since the symmetric key is generated by AES is generally faster than the asymmetric cryptographic algorithm. Then, we use symmetric key to encrypt the messages.

5.3 BA Protocol: "Server-initiated Byzantine Agreement Protocol" (SBAP)

To meet the characteristics of mobile environments in the BA problem, most of the communication and computation overhead must be fulfilled within in the agreement-servers. Therefore, only the agreement-server needs to exchange messages and compute the agreement value in our protocol. All messages in SBAP are encrypted by the symmetric key to ensure the security. There are three phases in SBAP; they are message-exchanging phase, decision-making phase and agreement-distribution phase. The protocol SBAP is shown in Figure 5-3.

5.3.1 The Number of Required Rounds of Message-Exchange

In the BA protocol, we use term "round" to compute the amount of messages exchange. The term "round" denotes the interval of message exchange between any pair of processors [19] [28] . Fischer and Lynch [19] indicated that $t+1$ ($t=\lfloor(n-1)/3\rfloor$) rounds are the minimum number of rounds required to get enough messages to achieve BA.

The network architecture by Fischer and Lynch [19] is flat architecture, but the network architecture in our system is a hierarchical architecture. In our protocol, only agreement-servers need to exchange messages in the message-exchanging phase. Therefore, the number of required rounds of message-exchange is z_m+1 ($z_m=\lfloor(z_n-1)/3\rfloor$). That is, SBAP can reduce the entire network transmission consumption

5.3.2 Message-Exchanging Phase

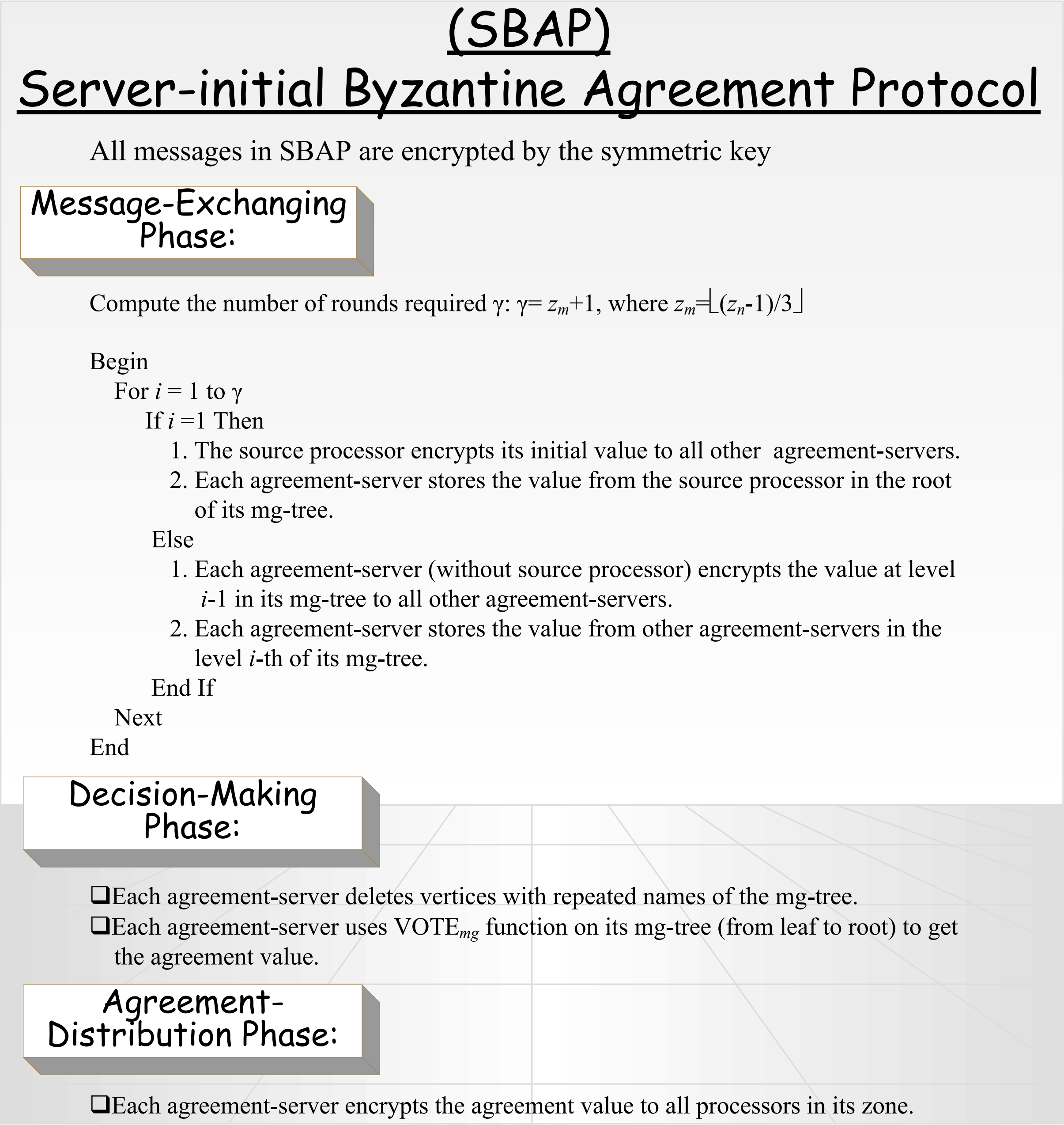
Each agreement-server computes the number of rounds required γ ($\gamma= z_m+1$, where $z_m=(z_n-1)/3$) at first. At first round of message-exchanging phase, only the source processor needs to encrypt its initial value to all other agreement-servers. Each agreement-server then stores the value from the source processor in the root of its mg-tree. At the $i\neq1$ round of message-exchanging phase, each agreement-server (without source processor) encrypts the value at level i-1 round in its mg-tree to all other agreement-servers. Each agreement-server then stores the value from other agreement-servers in the level *i*-th of its mg-tree.

5.3.3 Decision-Making Phase

After message-exchanging phase, each agreement-server deletes vertices with repeated names of mg-tree to avoid the repeated influence from faulty processors. Then, each agreement-server uses the VOTE$_{mg}$ function on its mg-tree from leaf to root to obtain the agreement value. The VOTE$_{mg}$ function is shown in Figure 5-4.

5.3.4 Agreement-Distribution Phase

Each agreement-server encrypts its agreement value to all processors in its zone. All fault-free processors (both stationary processors and mobile processors), which are managed by the fault-free agreement-server, can then reach a common agreement value. The value agreed upon by a faulty processor is ignored [28] .



(SBAP)

Server-initial Byzantine Agreement Protocol

All messages in SBAP are encrypted by the symmetric key

Message-Exchanging Phase:

```
Compute the number of rounds required $\gamma$: $\gamma = z_m+1$, where $z_m=\lfloor (z_n-1)/3 \rfloor$

Begin
   For $i$ = 1 to $\gamma$
       If $i$ =1 Then
           1. The source processor encrypts its initial value to all other  agreement-servers.
           2. Each agreement-server stores the value from the source processor in the root
              of its mg-tree.
       Else
           1. Each agreement-server (without source processor) encrypts the value at level
              $i$-1 in its mg-tree to all other agreement-servers.
           2. Each agreement-server stores the value from other agreement-servers in the
              level $i$-th of its mg-tree.
       End If
   Next
End
```

Decision-Making Phase:

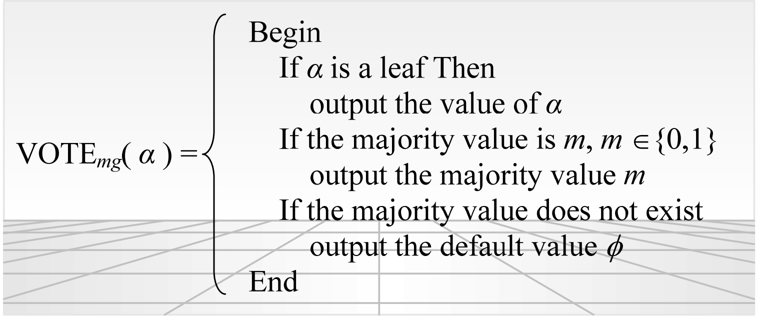
❑Each agreement-server deletes vertices with repeated names of the mg-tree.

❑Each agreement-server uses VOTE$_{mg}$ function on its mg-tree (from leaf to root) to get the agreement value.

Agreement-Distribution Phase:

❑Each agreement-server encrypts the agreement value to all processors in its zone.

Figure 5-3. The BA protocol Server-initial Byzantine Agreement Protocol (SBAP)



```
VOTE$_{mg}$( $\alpha$ ) =
Begin
   If $\alpha$ is a leaf Then
       output the value of $\alpha$
   If the majority value is $m$, $m \in \{0,1\}$
       output the majority value $m$
   If the majority value does not exist
       output the default value $\phi$
End
```

Figure 5-4. The VOTE$_{mg}$ Function

5.4 An Example of Reaching Byzantine Agreement

In this section, we present a short synopsis of the SBAP execution protocol. A combined wire/wireless network is shown in Figure 5-1. There are sixteen processors (including four agreement-servers, five stationary processors and seven mobile processors) falling into four zones. For example, there are three mobile processors (A_1, A_2 and A_3) in the zone A, and they are managed by agreement-server AS_A. The malicious faulty components are agreement-server AS_A, mobile processor C_3 and stationary processor D_1.

The source processor is the most important in the BA protocol. If the source processor has a malicious fault, it may send different initial values to different processors in the first round of message-exchanging phase. Therefore, the worst case BA problem is that the source processor has a malicious fault. If the BA protocol can solve the worst case, the BA problem can be solved in other cases. Hence, we suppose that the agreement-server AS_A is the source processor. To reach a common agreement value among all fault-free component in our example, the SBAP needs 2 ($\lfloor(4-1)/3\rfloor+1$) message-exchange rounds.

In the first round of message-exchanging phase, the source processor AS_A encrypts its initial value to all other agreement-servers in the network. Agreement-servers AS_B, AS_C, and AS_D then store the value from the source processor AS_A in the root of their mg-trees, as shown in Figure 5-5. In the second round of message-exchanging phase, each agreement-server (without source processor) encrypts the value at the root in its mg-tree to all other agreement-servers. The 2-level mg-tree of agreement-server AS_B in the second round of message-exchanging phase is shown in Figure 5-6. In the decision-making phase, each agreement-server deletes the vertices with repeated mg-tree names to avoid the repeated influence from faulty processors. In our example, there is no vertex with a repeated name. The VOTE_{mg} function is then used on its mg-tree from leaf to root to get the agreement value. For example, agreement-server AS_B computes VOTE(A) = (0,1,1) = 1 (VOTE(A) = (VOTE(AB), VOTE(AC), VOTE(AD))). An agreement value 1 is obtained. In the agreement-distribution phase, each agreement-server encrypts its agreement value to all processors in its zone. Therefore, agreement-server AS_B encrypts its agreement value 1 to processor B_1 and processor B2 in the zone B.

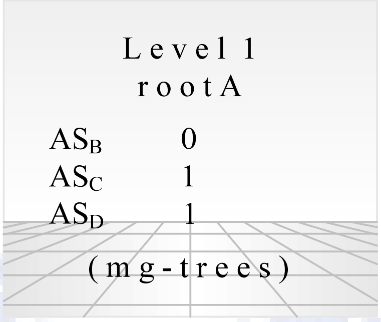


Figure 5-5. The mg-trees of each agreement-server in the first round of message-exchanging phase

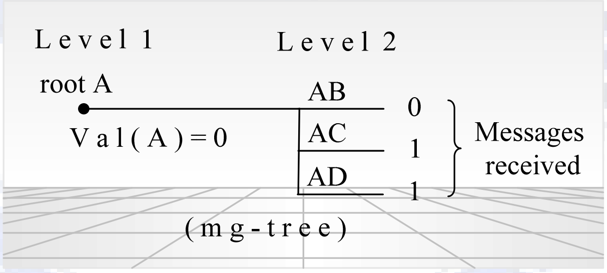


Figure 5-6. The 2-level mg-tree of agreement- server AS_B

5.5 The Correctness and Complexity of SBAP

If the value stored in vertex α of each fault-free agreement-server's mg-tree is identical, then the vertex α is called common [4] . When each fault-free agreement-server's has the common initial value from the source agreement-server in the root of its mg-tree, then an agreement is reached because the root is common. Thus, the constraints, (Agreement) and (Validity), can be rewritten as:

(BA_Agreement'): Root value is common.

(BA_Validity'): VOTE(α) = initial value of source agreement-server, for each fault-free agreement-server, if the source processor is fault-free.

To prove that a vertex is common, the term common frontier [4] is defined as follows: When every root-to-leaf path of the mg-tree contains a common vertex, the collection of the common vertices forms a common frontier. In other words, every fault-free agreement-server has the same messages collected in the common frontier if a common frontier does exist in a fault-free agreement-server's mg-tree. Subsequently, using the same function VOTE$_{mg}$ to compute the root value of the tree structure, every fault-free agreement-server can compute the same root value because the same input (the same collected messages in the common frontier) and the same computing function will produce the same output (the root value).

Lemma 5-1: All correct vertices of an mg-tree are common.

Proof: In the decision-making phase, all vertices with repeated names are deleted in an mg-tree. At level z_m+1 or above, the correct vertex α has at least 2z_m+1 children, out of which at least z_m+1 children are correct. The true values of these z_m+1 correct vertices are common, and the majority of the vertex values α are common. The correct vertex α is common in the mg-tree if the level of α is less then z_m+1. Consequently, all correct vertices of the mg-tree are common. ■

Lemma 5-2: The common frontier exists in the mg-tree.

Proof: There are z_m+1 vertices along each root-to-leaf path of an mg-tree in which the root is labeled by the source name, and the others are labeled by a sequence of agreement-server id. Since at most z_m agreement-server can fail, there is at least one correct vertex along each root-to-leaf path of the mg-tree. Using Lemma 5-1, the correct vertex is common and the common frontier exists in each fault-free agreement-server's mg-tree. ■

Lemma 5-3: Let α be a vertex, α is common if there is a common frontier in the sub-tree rooted at α.

Proof: If the height of α is 0 and the common frontier (α itself) exists, α is common. If the height of α is γ, the children of α are all in common under the induction hypothesis with the height of the children being γ-1. ■

Corollary 5-1: The root is common if the common frontier exists in the mg-tree.

Theorem 5-1: The root of a fault-free agreement-server's mg-tree is common.

Proof: Using Lemmas 3-1, 3-2, 3-3 and Corollary 5-1, the theorem is proved. ■

Theorem 5-2: Protocol SBAP solves the BA problem in a two-level combined wired/wireless network.

Proof: To prove this theorem, SBAP must meet the constraints (BA_Agreement') and (BA_Validity')

(BA_Agreement'): Root value is common. By Theorem 5-1, (BA_Agreement') is satisfied

(BA_Validity'): VOTE(α) = v for all fault-free agreement-servers, if the initial value of the source agreement-server is v_s say $v = v_s$.

Most agreement-servers are fault-free. The value of the correct vertices for all of the fault-free agreement-servers' mg-trees is v. Therefore, each correct vertex of the mg-tree is common (Lemma 5-1), and its true value is v. Using Theorem 5-1, this root is common. The computed value VOTE(α) = v is stored in the root for all the fault-free agreement-server. Therefore, (BA_Validity') is satisfied. ■

Theorem 5-3: SBAP requires z_m+1 rounds in the "message-exchanging phase" to solve the BA problem in a two-level combined wired/wireless network, and z_m+1 ($z_m=\lfloor(z_n-1)/3\rfloor$) is the minimum number of rounds in the "message-exchanging phase".

Proof: The "message-exchanging phase" is a time consuming phase. Fischer and Lynch [19] indicated that t+1 ($t=\lfloor(n-1)/3\rfloor$) rounds are the minimum number of rounds required to get enough messages to achieve BA. The network architecture of Fischer and Lynch [19] is flat architecture, but the network architecture of our system is hierarchical architecture. In our protocol, only agreement-servers need to exchange the messages in the message-exchanging phase, so the number of required rounds of message-exchange is z_m+1 ($z_m=\lfloor(z_n-1)/3\rfloor$). Thus, SBAP requires z_m+1 rounds, and this number is the minimum. ■

5.6 Conclusion

Combined wired/wireless networks have become popular because they have the advantages of both wired network (e.g., powerful computation ability, high bandwidth, reliability and so on.) and wireless network (e.g., mobility, quick deployment and so on). Previous BA protocols [28] [37] [51] [60] [70] , were not applicable for combined wired/wireless networks. In this paper, we revisit the BA problem over a combined wired/wireless network with malicious faulty processors and use a hierarchical architecture to reduce the communication overhead.

Base on the preceding discussion, the protocol SBAP have the following features:

- Most of the communication and computation overhead are fulfilled within in agreement-servers. (To meet the characteristics of mobile environments, most of the communication and computation overhead must be fulfilled within in the agreement-servers.)
- SBAP can reduce the number of message-exchange rounds (SBAP uses the

hierarchical model concept to reduce the number of message-exchange rounds.)

- SBAP can reach a common agreement with malicious faulty processors in two-level combined wired/wireless networks. (By Theorem 5-2.)
- The number of message-exchange rounds for SBAP is the minimum. (By Theorem 5-3.)

Chapter 6

Client-initiated Consensus Protocol for Combined Wired/Wireless Networks

In chapter 5, we proposed a server-initiated BA protocol, SBAP, to solve the BA problem with malicious faulty processors in a combined wired/wireless network. In this chapter, we consider another related problem: Consensus problem. Moreover, malicious fault assumption with processors grows into the dual failure mode (both dormant fault and malicious fault) on both processors and communication links.

That is, we introduce our approach to solve the Consensus problem with dormant/malicious faulty processors/communication links in a combined wired/wireless network in this chapter.

6.1 The Conditions for Consensus Problem in Combined Wired/Wireless Network

To design a Consensus protocol, certain conditions must be taken into account. They are the system model, properties of the Consensus problem and the constraint.

6.1.1 System Model

In recent years, the bandwidth and quality of wireless networks has been drastically improved. Therefore, wireless network has become more and more popular [9] , resulting in the development of nowadays network from wired or wireless network to combined wired/wireless network.

We know that the communication overhead of the Consensus protocol is inherently large [19] . Previous Consensus protocols were designed for flat networks [52] [70] In a flat network, all processors undertake equal responsibility and all messages must propagate globally throughout the network. This makes the previous Consensus protocols inefficient. In this study, we use a hierarchical concept to reduce the communication overhead.

In this chapter, the Consensus problem is considered in a combined wired/wireless network with fallible components (processor and communication link). The failure type of a fallible component may be dormant fault or malicious fault. Figure 6-1 shows an example of the two-level combined wired/wireless network. There are six consensus-servers, six stationary processors and nine mobile processors. The network is divided into six zones by six consensus-servers. Each consensus-server manages a zone's processors. For example, consensus-server CS_A manages processor A_1, A_2 and A_3 in the zone A. The assumptions and parameters of the system are listed in chapter 3.2.2.

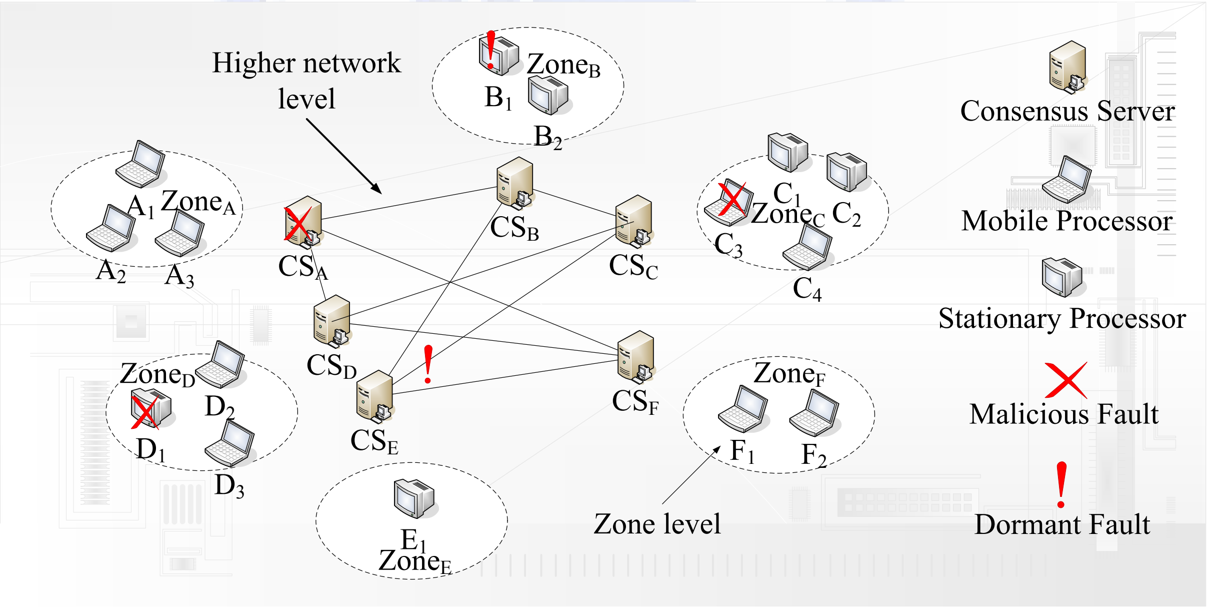


Figure 6-1. Two-level combined wired/wireless network

6.1.2 Properties of the Consensus Problem

Using the hierarchical concept, each processor in the zone is managed by its consensus-server. Hence, two properties of the Consensus problem in two-level combined wired/wireless network are modified as follows:

(Consensus_Agreement): All fault-free processors managed by fault-free consensus-server agree on a common value;

(Consensus_Validity): If the initial value of all consensus-server is v_i, then all fault-free processors managed by the fault-free consensus-server shall agree on v_i.

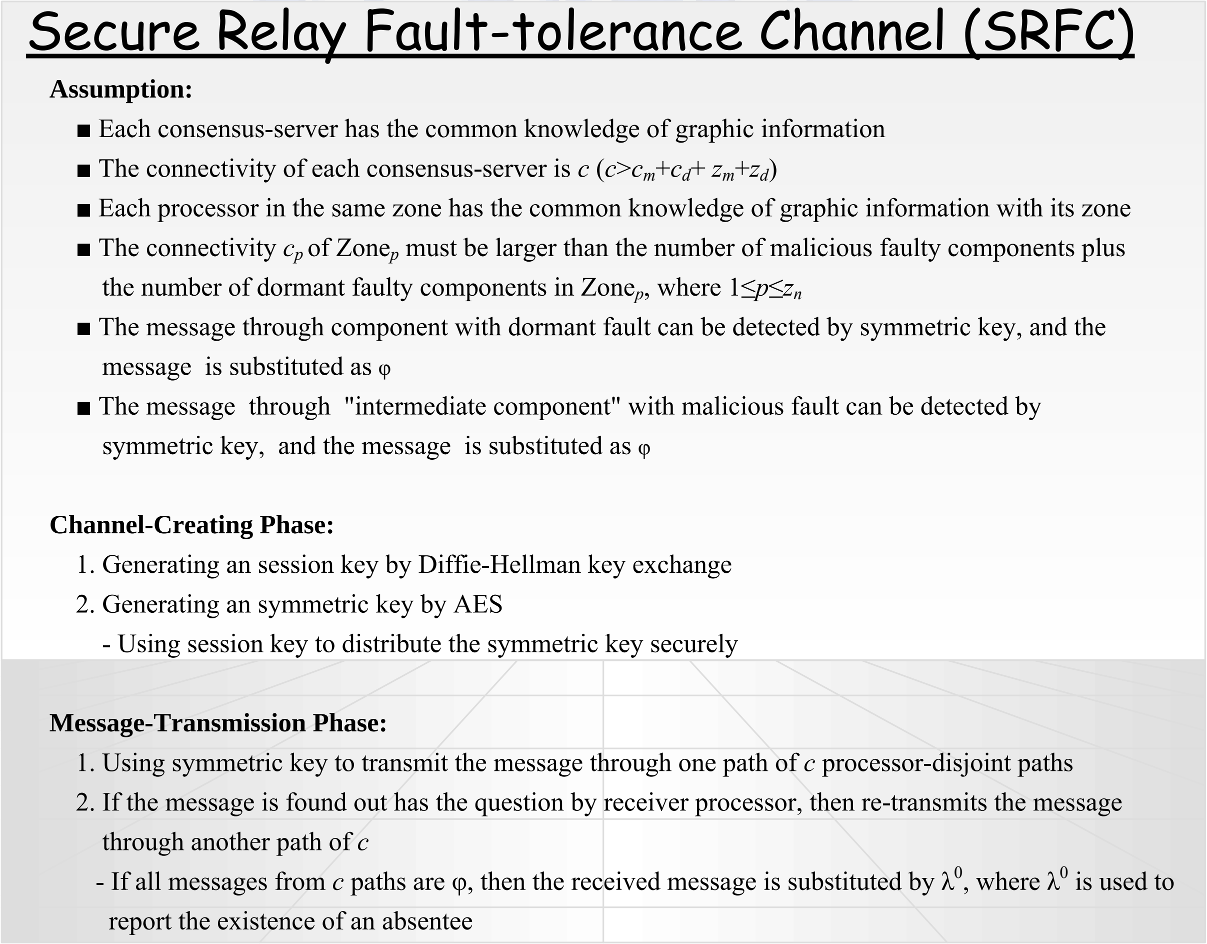
6.1.3 Constraint

In the Consensus problem, the number of faulty processors allowed in the network depends on the total number of processors. Meyer and Pradhan [33] indicates the constraint of Consensus problem with malicious faulty processors and dormant faulty processors is $n > 3p_m+p_d$. Afterward, Siu et al. [50] finds that the correct constraint should be $n > \lfloor(n-1)/3\rfloor+2p_m+p_d$.

The network architectures of Meyer and Pradhan [33] and Siu et al. [50] are flat. All processors need to exchange the messages in the message-exchanging phase. In our Consensus protocol, the network architecture is hierarchical, and only the consensus-server needs to exchange the messages in the message-exchanging phase. Hence, the constraint of our model is $z_n > \lfloor(z_n-1)/3\rfloor+2z_m+z_d$.

6.2 Transmission Protocol: "Secure Relay Fault-tolerance Channel" (SRFC)

Energy consumption is a major performance metric for mobile processors [48] . If the power consumption is low, the battery lifetime will be longer. In this section, a transmission protocol "Secure Relay Fault-tolerance Channel" (SRFC) is proposed. SRFC can remove the influence from the faulty intermediate component and reduces power consumption to provide a secure communication channel between sender and receiver. The transmission protocol SRFC is shown in Figure 6-2.



Secure Relay Fault-tolerance Channel (SRFC)

Assumption:

- ■ Each consensus-server has the common knowledge of graphic information
- ■ The connectivity of each consensus-server is c ($c>c_m+c_d+ z_m+z_d$)
- ■ Each processor in the same zone has the common knowledge of graphic information with its zone
- ■ The connectivity c_p of Zone$_p$ must be larger than the number of malicious faulty components plus the number of dormant faulty components in Zone$_p$, where $1\leq p\leq z_n$
- ■ The message through component with dormant fault can be detected by symmetric key, and the message is substituted as φ
- ■ The message through "intermediate component" with malicious fault can be detected by symmetric key, and the message is substituted as φ

Channel-Creating Phase:

1. Generating an session key by Diffie-Hellman key exchange
2. Generating an symmetric key by AES
   - Using session key to distribute the symmetric key securely

Message-Transmission Phase:

1. Using symmetric key to transmit the message through one path of c processor-disjoint paths
2. If the message is found out has the question by receiver processor, then re-transmits the message through another path of c
   - If all messages from c paths are φ, then the received message is substituted by λ^0, where λ^0 is used to report the existence of an absentee

Figure 6-2. Secure Relay Fault-tolerance Channel (SRFC)

6.2.1 The Connectivity Constraint

In Meyer and Pradhan [33] , the fallible components are dormant/malicious faulty processors. Meyer and Pradhan [33] used a time-out mechanism to detect a dormant faulty processor. However, the time-out mechanism cannot detect a malicious faulty intermediate component. To avoid the majority value from being dominated by malicious faulty intermediate components, the connectivity constraint in the network by Meyer and Pradhan [33] is c' ($c' > 2p_m + p_d$, where p_m is the maximum number of malicious faulty processors allowed and p_d is the maximum number of dormant faulty processors allowed). In Siu, Chin and Yang [50] , the fallible components are dormant/malicious faulty processors and dormant/malicious faulty communication links. Siu, Chin and Yang [50] also used a time-out mechanism to detect a dormant faulty component. Thus, the connectivity constraint in the network by Siu, Chin and Yang [50] is c'' ($c'' > 2p_m + p_d + 2(l_m + l_d)$, where p_m is the maximum number of malicious faulty processors allowed, p_d is the maximum number of dormant faulty processors allowed, l_m is the maximum number of malicious faulty communication links allowed and l_d is the maximum number of dormant faulty communication links allowed).

In SRFC, the network connectivity constraint is improved. For the higher network level, the connectivity of each consensus-server is c ($c > c_m + c_d + z_m + z_d$ where c_m is the maximum number of malicious faulty communication link allowed, c_d is the maximum number of dormant faulty communication links allowed, z_m is the maximum number of malicious faulty consensus-servers allowed and z_d is the maximum number of dormant faulty consensus-servers allowed). For the zone level, the connectivity c_p of Zone$_p$ must be larger than the number of malicious faulty components (processors and communication links) plus the number of dormant faulty components in Zone$_p$, where $1 \leq p \leq z_n$. Because a symmetric key is used, the receiver processor can detect a message that is influenced by dormant and malicious faulty intermediate components.

6.2.2 Four Cases of Fault Handling

We classify the fault (or attack) that may take place in a transmission process into four cases:

1. Sender with dormant fault or intermediate component with dormant fault
2. Intermediate component with malicious fault
3. Sender with malicious fault
4. Receiver with dormant fault or receiver with malicious fault

Our protocol SRFC can deal with Cases 1 and Case 2. For Case 1, a message sent through a dormant faulty component cannot be reconstructed by the symmetric key. In Case 2, we can detect that the message is false using the symmetric key. In Case 3, our SRFC cannot detect if the message is correct or not. Because the sender has the symmetric key, the sender has control over the message. Case 3 can be solved using our Consensus protocol CCP. The detailed description of CCP is presented in chapter 6.3. For Case 4, because the receiver is a faulty component, we do not care the message received from the faulty receiver.

If the network connectivity is c, we can determine c processor-disjoint paths between the sender and receiver. These c processor-disjoint paths can be predefined [68] . An example of c(c=3) processor-disjoint path between CS_B and CS_C is shown in Figure 6-3. In Path 2 there is a dormant faulty communication link between CS_E and CS_C. Hence, the message is influenced by the dormant faulty component (Case 1). CS_C can detect this problem using the symmetric key. In Path 3 there is a malicious faulty component CS_A (Case 2), CS_C also can detect this problem using the symmetric key. In Path 1, there is no faulty intermediate component between CS_B and CS_C, CS_C can receive the message from CS_B without faulty influence. That is, if $c>c_m+c_d+z_m+z_d$ (3>0+1+1+0, 3>2), we can ensure that the receiver can receive the message without influence from faulty intermediate component.

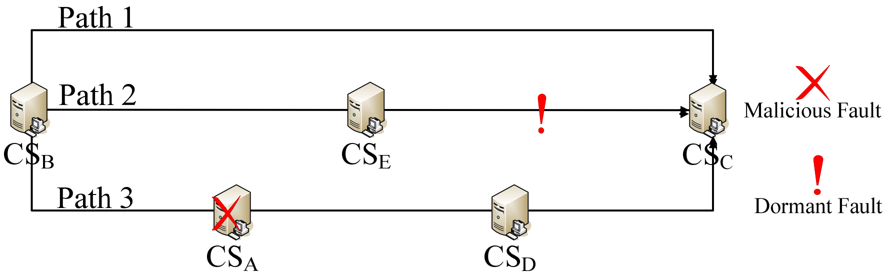


Figure 6-3. The *c* disjoint paths between CS$_B$ and CS$_C$, where *c*=3

To reduce power consumption, each sender only transmits one copy of the message through one path of *c* paths. If the receiver detects that the message is false, it then re-transmits the message through another path in *c* processor-disjoint paths. If all messages from *c* paths are φ, then the received message is substituted by λ^0. λ^0 is used to report the existence of an absentee.

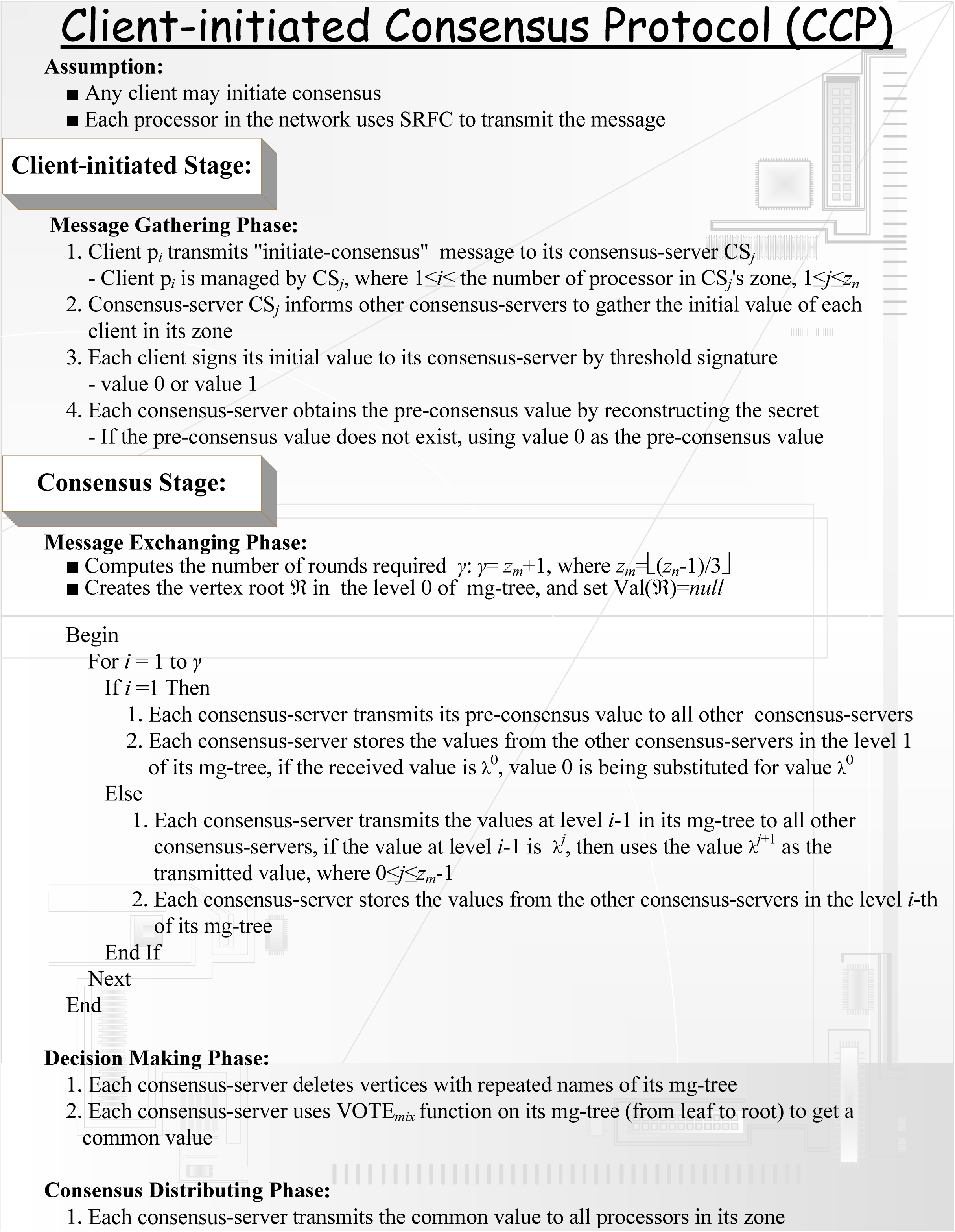
That is, SRFC can remove the influence from dormant/malicious faulty intermediate components and the influence from dormant faulty sender. SRFC is an efficient transmission protocol which reduces the computation time and power consumption to provide a secure communication channel.

6.3 Consensus Protocol: "Client-initiated Consensus Protocol" (CCP)

In this section, we would like to focus our attention on the proposed Consensus protocol "Client-initiated Consensus Protocol" (CCP). To meet the characteristics of Consensus problem in combined wired/wireless networks, most of the communications and computation overhead must be fulfilled within by the consensus-servers. Therefore, only consensus-servers need to exchange messages and compute the common value in CCP. Furthermore, all messages in CCP are transmitted by SRFC. There are two stages in CCP, namely the Client-initiated Stage and the Consensus Stage. The Consensus protocol CCP is shown in Figure 6-4.

6.3.1 Client-initiated Stage

The purpose of the Client-initiated Stage is to collect the initial value from each client and compute the pre-consensus value for each consensus-server. Any client may initiate Consensus in the network. When a client p_i wants to initiate a Consensus, it transmits the “initiate consensus” message to its consensus-server CS$_j$. (Client p_i is managed by CS$_j$, where 1≤i≤ the number of processor in CS$_j$'s zone, 1≤j≤z_n.) The consensus-server CS$_j$ then informs other consensus-servers to gather the initial value from each client in its zone. Each consensus-server can obtain the pre-consensus value by threshold signature if it collects more than half of the same value in its zone.



Client-initiated Consensus Protocol (CCP)

Assumption:

- ■ Any client may initiate consensus
- ■ Each processor in the network uses SRFC to transmit the message

Client-initiated Stage:

Message Gathering Phase:

1. Client p_i transmits "initiate-consensus" message to its consensus-server CS_j
   - Client p_i is managed by CS_j, where $1 \leq i \leq$ the number of processor in CS_j's zone, $1 \leq j \leq z_n$
2. Consensus-server CS_j informs other consensus-servers to gather the initial value of each client in its zone
3. Each client signs its initial value to its consensus-server by threshold signature
   - value 0 or value 1
4. Each consensus-server obtains the pre-consensus value by reconstructing the secret
   - If the pre-consensus value does not exist, using value 0 as the pre-consensus value

Consensus Stage:

Message Exchanging Phase:

- ■ Computes the number of rounds required γ: $\gamma = z_m + 1$, where $z_m = \lfloor (z_n - 1)/3 \rfloor$
- ■ Creates the vertex root $\Re$ in the level 0 of mg-tree, and set Val($\Re$)=*null*

```
Begin
   For $i = 1$ to $\gamma$
      If $i = 1$ Then
         1. Each consensus-server transmits its pre-consensus value to all other consensus-servers
         2. Each consensus-server stores the values from the other consensus-servers in the level 1
            of its mg-tree, if the received value is $\lambda^0$, value 0 is being substituted for value $\lambda^0$
      Else
         1. Each consensus-server transmits the values at level $i$-1 in its mg-tree to all other
            consensus-servers, if the value at level $i$-1 is $\lambda^j$, then uses the value $\lambda^{j+1}$ as the
            transmitted value, where $0 \leq j \leq z_m - 1$
         2. Each consensus-server stores the values from the other consensus-servers in the level $i$-th
            of its mg-tree
      End If
   Next
End
```

Decision Making Phase:

1. Each consensus-server deletes vertices with repeated names of its mg-tree
2. Each consensus-server uses $VOTE_{mix}$ function on its mg-tree (from leaf to root) to get a common value

Consensus Distributing Phase:

1. Each consensus-server transmits the common value to all processors in its zone

Figure 6-4. Client-initiated Consensus Protocol (CCP)

6.3.2 Consensus Stage

The purpose of the Consensus Stage is to compute a common value. There are three phases in the Consensus stage; including the message-exchanging phase, the decision-making phase and the consensus-distribution phase. In the message-exchanging phase, each consensus-server computes the number of rounds required at first. Then, each consensus-server creates the vertex $\Re$ in the level 0 of its mg-tree, and set Val($\Re$)=*null*.

6.3.2.1 The Number of Required Rounds in Message-Exchanging phase

In the Consensus protocol, we also use term "round" to compute the number of messages exchanged. In our protocol, only consensus-servers need to exchange the messages in the message-exchanging phase. So the number of required rounds in the message-exchanging phase is z_m+1 ($z_m=\lfloor(z_n-1)/3\rfloor$).

6.3.2.2 Message-Exchanging Phase

In the first round of the message-exchanging phase, each consensus-server transmits its pre-consensus value to all other consensus-servers. Each consensus-server then stores the values from the other consensus-servers in level 1 of its mg-tree. If the received value is λ^0, value 0 is substituted for value λ^0.

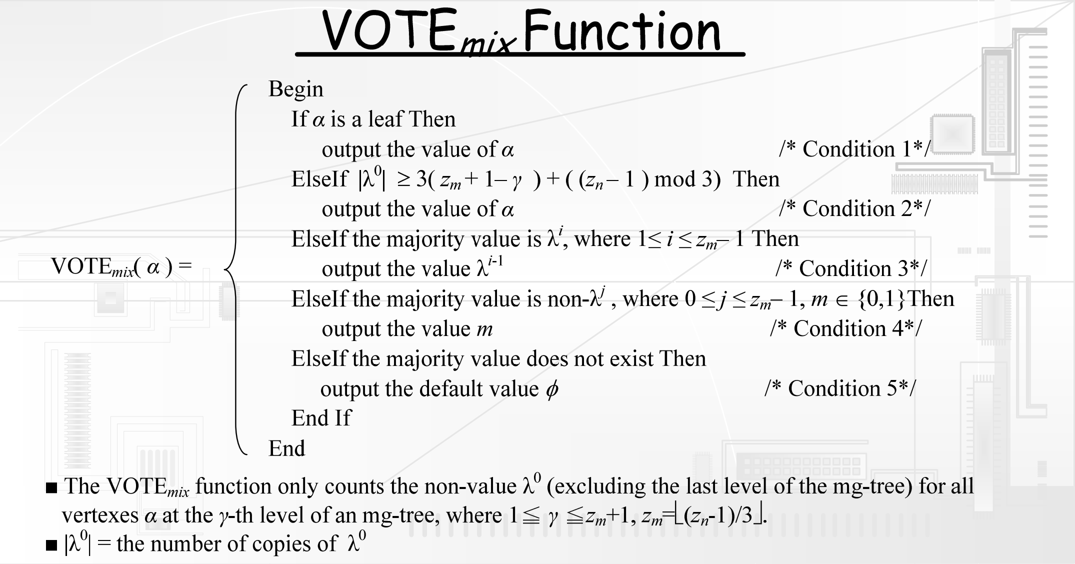
In the $i\neq1$ round of message-exchanging phase, each consensus-server transmits the values at level i-1 round in its mg-tree to all other consensus-servers.

6.3.2.3 Decision-Making Phase

After the message-exchanging phase, each consensus-server deletes vertices with repeated names of mg-tree to avoid the repeated influence from faulty consensus-servers. Then, each consensus-server uses the VOTE$_{mix}$ function on its mg-tree from leaf to root to obtain the common value. The VOTE$_{mix}$ function is shown in Figure 6-5. Conditions 1, 4 and 5 are similar to convention majority vote [28] . Condition 2 is used to deal with the dual failure mode (where both dormant fault and malicious fault exist). Condition 3 is used to describe the existence of an absentee.

6.3.2.4 Consensus-Distribution Phase

Each consensus-server transmits the common value to all processors in its zone. All fault-free processors (both stationary processors and mobile processors), which are managed by the fault-free consensus-server, can obtain a common value. The value agreed upon by a processor, which is managed by faulty consensus-server, is ignored [28] .



VOTE$_{mix}$ Function

```
VOTE_mix( $\alpha$ ) =
Begin
  If $\alpha$ is a leaf Then
     output the value of $\alpha$                                   /* Condition 1*/
  ElseIf $|\lambda^0| \geq 3(z_m + 1 - \gamma) + ((z_n - 1) \bmod 3)$ Then
     output the value of $\alpha$                                   /* Condition 2*/
  ElseIf the majority value is $\lambda^i$, where $1 \leq i \leq z_m - 1$ Then
     output the value $\lambda^{i-1}$                                /* Condition 3*/
  ElseIf the majority value is non-$\lambda^j$ , where $0 \leq j \leq z_m - 1$, $m \in \{0,1\}$Then
     output the value $m$                                        /* Condition 4*/
  ElseIf the majority value does not exist Then
     output the default value $\phi$                              /* Condition 5*/
  End If
End
```

- ■ The VOTE$_{mix}$ function only counts the non-value λ^0 (excluding the last level of the mg-tree) for all vertexes α at the γ-th level of an mg-tree, where $1 \leqq \gamma \leqq z_m+1$, $z_m = \lfloor (z_n-1)/3 \rfloor$.
- ■ $|\lambda^0|$ = the number of copies of λ^0

Figure 6-5. The VOTE$_{mix}$ Function

6.4 An Example of Reaching Consensus

In this section, we give an example of executing SRFC and CCP. A two-level combined network is shown in Figure 6-1. The dormant faulty components are B_1 and L_{CE}. The malicious faulty components are CS_A, D_1 and C_3.

6.4.1 Client-initiated Stage

B_2 wants to initiate a Consensus. Hence, B_2 creates a secure communication channel between CS_B by SRFC. B_2 then transmits the "initiate-consensus" message to CS_B through processor-disjoint path created by SRFC. After CS_B receives the "initiate-consensus" message, CS_B creates secure communication channels to all other consensus-servers by SRFC and then informs all consensus-servers to gather the initial value from each client in its zone. The initial value of each client is shown as follows.

Client ID:	A_1	A_2	A_3	B_1	B_2	C_1	C_2	C_3	C_4	D_1	D_2	D_3	E_1	F_1	F_2
Initial value:	0	1	0	λ^0	1	1	1	1	1	1	0	1	0	0	0

Each consensus-server then obtains the pre-consensus value using the threshold signature. For example, there are two processors in $Zone_B$, one is B_1 which is a dormant faulty processor, and another is B_2 which is a fault-free processor. Hence, CS_B can detect that the message from B_1 is influenced by a dormant faulty component. After B_1 and B_2 sign its initial value to CS_B, CS_B can obtain the pre-consensus value 1 (the number of value 1 is greater than or equal to half of the number of processor in $Zone_B$, λ^0 is ignore). The pre-consensus value of each consensus-server is shown as follows.

Consensus-Server ID:	CS_A	CS_B	CS_C	CS_D	CS_E	CS_F
Pre-consensus value:	0	1	1	1	0	0

6.4.2 Consensus Stage

In the message-exchanging phase, we first compute the number of rounds required $\gamma=2$ ($\gamma=\lfloor(6-1)/3\rfloor+1$). Then, we creates the vertex $\Re$ in the level 0 of its mg-tree, and set Val($\Re$)=*null*. In the first round of the message-exchanging phase, each consensus-server transmits its pre-consensus value to all other consensus-servers by SRFC. Each consensus-server then stores the values from other consensus-servers in the level 1 of its mg-tree. Since CS_A is a malicious faulty processor, CS_A may transmit different messages to different consensus-server to prevent the fault-free consensus-server from reaching a common value. The messages transmitted by CS_A in the first round of message-exchanging phase are shown in Figure 6-6. An mg-tree of CS_B after the first round of message-exchanging phase is shown in Figure 6-7.

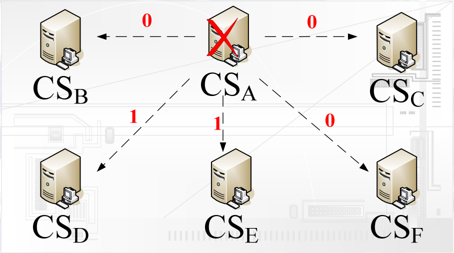


Figure 6-6. CS_A transmits different message to different consensus-server

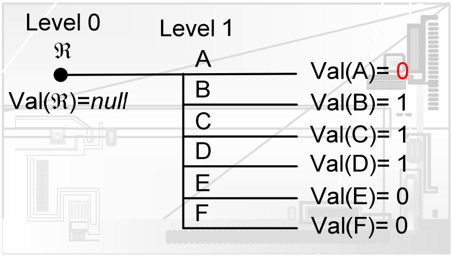


Figure 6-7. An one-level mg-tree of CS_B

In the second round of the message-exchanging phase, each consensus-server transmits the value at level 1 in its mg-tree to all other consensus-servers, and stores the value from other consensus-servers in the level 2 of its mg-tree. An mg-tree of CS_B after the second round of the message-exchanging phase is shown in Figure 6-8.

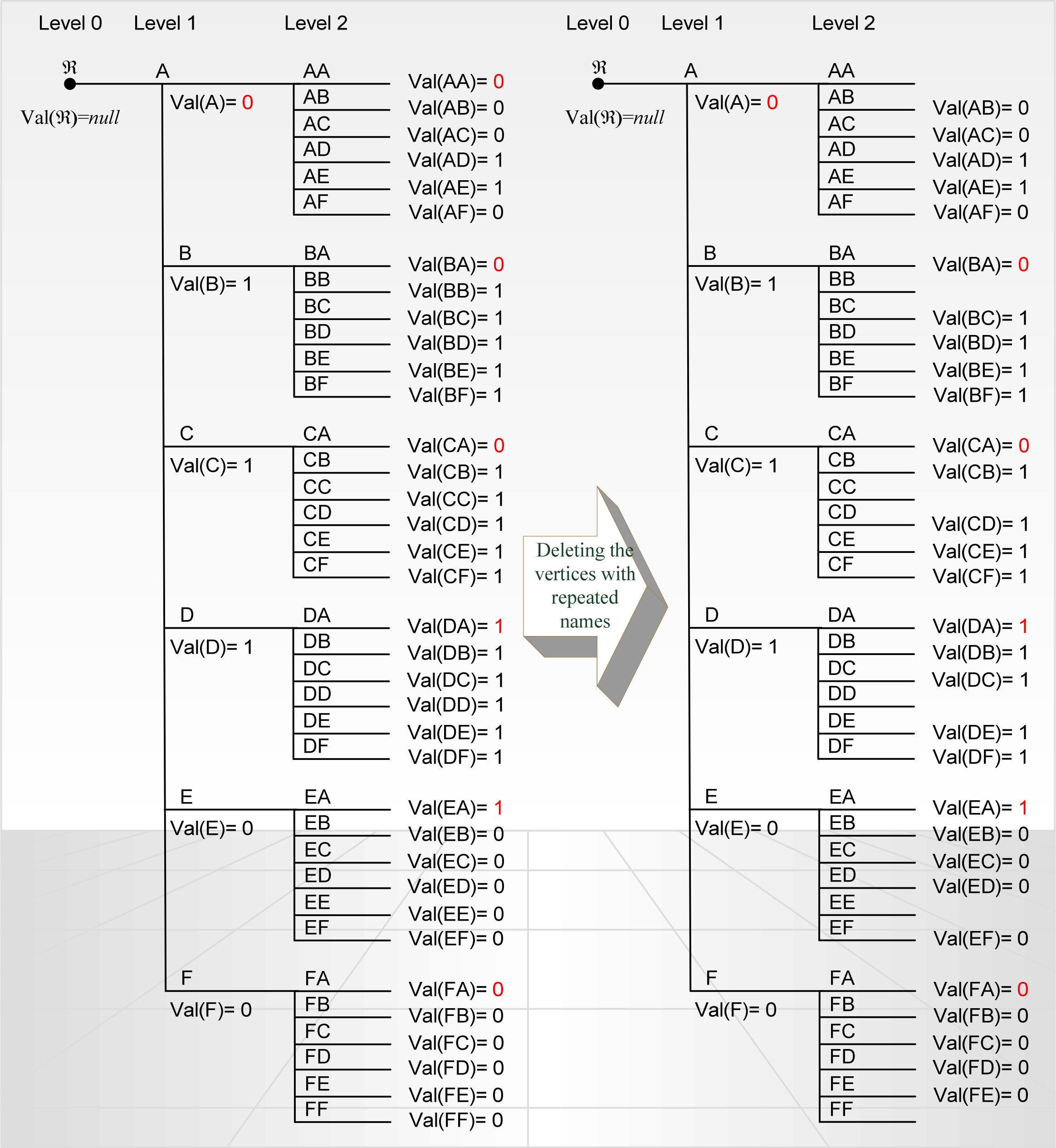


Figure 6-8. An mg-tree of CS_B after the second round of the message-exchanging phase

Figure 6-9. An mg-tree of CS_B without repeated name vertices

In the decision-making phase, the vertices with repeated names of mg-tree are deleted by each consensus-serer. An mg-tree of CS$_B$ without repeated name vertices is shown in Figure 6-9. Each consensus-server then uses the VOTE$_{mix}$ function on its mg-tree (from leaf to root) to compute the common value. For example, CS$_B$ can obtain the common value ϕ by VOTE$_{mix}$. VOTE$_{mix}$($\mathfrak{R}$) = ((0,0,1,1,0), (0,1,1,1,1), (0,1,1,1,1), (1,1,1,1,1), (1,0,0,0,0), (0,0,0,0,0)) = (0,1,1,1,0,0) = ϕ.

6.5 The Correctness and Complexity of CCP

The goal of CCP is to enable all fault-free consensus-server to reach a common value to solve the Consensus problem in a combined wired/wireless network. To prove the correctness of our protocol CCP, a vertex $\mathfrak{R}$ is called common [4] if each fault-free consensus-server has the same value for $\mathfrak{R}$. That is, if vertex $\mathfrak{R}$ is common, then the value stored in vertex $\mathfrak{R}$ of each fault-free consensus-server's mg-tree is identical.

Lemma 6-1: All correct vertices of an mg-tree are common after function VOTE$_{mix}$ is applied to mg-tree.

Proof: In the decision-making phase, all vertices with repeated names are deleted in an mg-tree. At level z_m+1 or above, the correct vertex α has at least 2z_m+1 children, and out of which at least z_m+1 children are correct. The true values of these z_m+1 correct vertices are common, and the majority of the vertex value α is common. The correct vertex α is common in the mg-tree if the level of α is less then z_m+1. Consequently, all correct vertices of the mg-tree are common. ■

Lemma 6-2: The common frontier exists in the mg-tree.

Proof: By definition, an mg-tree is a tree of level z_m+1. There are z_m+1 vertices along each root-to-leaf path of an mg-tree. Since at most z_m consensus-servers can fail, there is at least one correct vertex along each root-to-leaf path of the mg-tree. Using Lemma 6-1, the correct vertex is common and the common frontier exists in each fault-free consensus-server's mg-tree. ■

Lemma 6-3: Let α be a vertex, α is common if there is a common frontier in the sub-tree rooted at α.

Proof: If the height of α is 0 and the common frontier (α itself) exists, α is common. If the height of α is γ, the children of α are all in common under the induction hypothesis with the height of the children being γ-1. ■

Corollary 6-1: The value of root $\Re$ is common if the common frontier exists in the mg-tree.

Theorem 6-1: The value of root $\Re$ of a fault-free consensus-server's mg-tree is common.

Proof: Using Lemmas 4-1, 4-2, 4-3 and Corollary 6-1, the theorem is proved. ■

Theorem 6-2: Protocol CCP solves the Consensus problem in a two-level combined wired/wireless network**.**

Proof: To prove this theorem, CCP must meet the constraints (Consensus') and (Validity')

(Consensus_Agreement): Root value is common.

By Theorem 6-1, (Consensus_Agreement') is satisfied.

(Consensus_Validity): VOTE(α) = v for all fault-free consensus-servers, if the initial value of all consensus-server is v_s say $v = v_s$.

Most consensus-servers are fault-free. The value of the correct vertices for all of the fault-free consensus-servers' mg-trees is v. Therefore, each correct vertex of the mg-tree is common (Lemma 6-1), and its true value is v. Using Theorem 6-1, this root is common. The computed value VOTE(α) = v is stored in the root for all the fault-free consensus-server. Therefore, (Validity') is satisfied. ■

Theorem 6-3: CCP requires z_m+1 rounds in the message-exchanging phase to solve the Consensus problem in a two-level combined wired/wireless network, and z_m+1 ($z_m=\lfloor(z_n-1)/3)\rfloor$) is the minimum number of rounds in the "message-exchanging phase"**.**

Proof: In the Consensus protocol, we use term "round" to compute the number of messages exchanged. A round is defined as follows: (1) sending messages to any set of nodes, (2) receiving messages, and (3) processing the messages locally [19] . The "message-exchanging phase" is a time consuming phase. Fischer and Lynch [19] indicated that t+1 ($t=\lfloor(n-1)/3\rfloor$) rounds are the minimum number of rounds required to get enough messages to achieve Consensus. The network architecture of Fischer and Lynch [19] is a flat architecture, but the network architecture of our system is two-level architecture. In our protocol, only consensus-servers need to exchange the messages in the message-exchanging phase, so the number of required rounds of message-exchange is z_m+1 ($z_m=\lfloor(z_n-1)/3\rfloor$). Thus, CCP requires z_m+1 rounds, and this number is the minimum. ■

6.6 Conclusion

Three motives are combined in this dissertation on the Consensus problem in combined wired/wireless network. First, most networks today are combined wired/wireless networks. Extant Consensus protocols are not applicable to combined wired/wireless networks. Hence, we proposed the protocol CCP to solve the Consensus problem in combined wired/wireless network. Second, the limited resources have made the computation ability of mobile processors often weaker than that of stationary processors. The proposed SRFC provides an efficient and secure communication channel. Third, the communication overhead of the

Consensus protocol is inherently large. We used the hierarchical concept in CCP to reduce the large amount of communication overheads. For example, if there are 128 processors that fall into 8 zones, the protocols designed for flat network need 43 ($\lfloor(n\text{-}1)/3\rfloor+1$) rounds in the message-exchanging phase to reach a common value. However, CCP only needs 3 ($\lfloor(z_n\text{-}1)/3\rfloor+1$) rounds in the message-exchanging phase to reach a common value. Therefore, CCP is more efficient than the previous protocols when the network is logically divided into hierarchical architecture. Table 6-1 shows some instances of the number of rounds required for flat network and two-level network. Smaller number of zone is preferred since the number of rounds required in the message-exchanging phase is smaller.

That is, we solved the Consensus problem with dual failure mode (both dormant and malicious fault) on both processors and communication links in the combined wired/wireless network. CCP requires only z_m+1 rounds (minimum number of rounds) in the message-exchanging phase which is optimal for all fault-free processors managed by fault-free consensus-servers to reach a common value.

Table 6-1 Some instances of the number of rounds required for various Consensus protocols

	The number of rounds required in message-exchanging phase			
		n=128, z_n=32	n=128, z_n=16	n=128, z_n=8
Flat Network	t+1, $t=\lfloor(n\text{-}1)/3\rfloor$	43	43	43
Two-Level Network	z_m+1, $z_m=\lfloor(z_n\text{-}1)/3\rfloor$	11	6	3

n: the number of processors in the underlying network and $n\geq4$.

z_n: the number of zones in the underlying network and $z_n\geq4$.

Chapter 7

Fault Diagnosis Agreement

In this chapter, we introduce our approach to solve the FDA problem in a combined wired/wireless network. The proposed protocol called Adaptive Fault Diagnosis Agreement Protocol (AFDA). AFDA is an adaptive FDA protocol. AFDA not only can solve the FDA problem in combined wired/wireless network, but also AFDA can solve the FDA problem in other networks. For example, if AFDA uses the evidence gathered from the BA protocol MAHAP, the malicious faulty processors, *away processors* and *return processors* can be detected/located in MANET. If AFDA uses the evidence gathered from the BA protocol SBAP, the the malicious faulty agreement-servers, *away processors* and *return processors* can be detected/located in combined wired/wireless network.

7.1 Proposed Protocol: "Adaptive Fault Diagnosis Agreement Protocol" (AFDA)

There are three phases in the AFDA: message-collection phase, fault-diagnosis phase and re-configuration phase. The message-collection phase is used to collect ic-trees of all processors/agreement-servers which executing the BA protocol MAHAP/SBAP (depends on the network). In order to ensure that the fault diagnosis result from each fault-free processor/agreement-server is the same in MANET/combined wired/wireless network, each fault-free processor/agreement-server should collect the same evidence. Thus, AFDA collects ic-trees of all processors/agreement-server by using MAHAP/SBAP (depends on the network). The fault-diagnosis phase is used to detect/locate *away processors*, *return processors*, and malicious faulty processors/agreement-servers. The set of *MFN* is used to record malicious faulty processors/agreement-servers, the set of *AN* is used to record processors which have ever moved away from the network, and the set of *RN* is used to record *return processors*. The re-configuration phase is used to re-configure the network by isolating malicious faulty processors/agreement-servers and away processors. The AFDA protocol is shown in Figure 7-1.

7.1.1 Message-Collection Phase

The goal of the message-collection phase is to collect ic-trees of all processors/agreement-servers which executing the BA protocol MAHAP/SBAP (except *return processors* and *away processors*) as evidence. In order to ensure that the fault diagnosis result from each fault-free processor/agreement-server is the same, each fault-free processor/agreement-server should collect the same evidence. Therefore, in the message-collection phase of AFDA, each processor/agreement-server (except *return processors* and *away processors*) uses MAHAP/SBAP (depends on the network) to distribute its ic-tree to all processors/agreement-servers. Then processor/agreement-server stores the other processors'/agreement-servers' ic-trees to construct the set of ***IC-trees*** =[ic-tree$_{s,}$ ic-tree$_{a}$, ic-tree$_{b}$ …, ic-tree$_{\ddot{\upsilon}}$] , where $\ddot{\upsilon}$ is the last Processor/agreement-server id in the network by alphabetical order. By using MAHAP/SBAP, we can ensure that fault-free processor/agreement-server collects the same set of ***IC-trees*** (the common set of ***IC-trees*****).** The detail description about how to collect the common set of ***IC-trees*** is shown in Lemma 7-1-1 and 6-2-1.

7.1.2 Fault-Diagnosis Phase

The goal of the fault-diagnosis phase is to detect/locate *away processors*, *return processors* and malicious faulty processors/agreement-servers. Each processor/agreement-server maintains the set of *MFN*, *AN* and *RN* in the fault-diagnosis phase. The set of *MFN* is used to record malicious faulty processors/agreement-servers, the set of *AN* is used to record processors which have ever moved away from the network and the set of *RN* is used to record *return processors*. Each processor/agreement-server examines the common set of ***IC-trees*** in a top-down and level by level sequence by step2 in the fault-diagnosis phase of AFDA.

7.1.2.1 Detect/Locate away processors and return processors

Some of processor has mobility in the network, and *away processors* can be detected by the system. If Processor ρ has ever been an *away processor*, each processor sets $AN = AN \cup \{\rho\}$, where ρ is Processor id. Moreover, *return processors* can also be detected, so each processor can also record the *return processor* in the set of *RN*. If Processor Ъ is a *return processor*, then *RN*=*RN* ∪ {Ъ}, where Ъ is Processor id.

7.1.2.2 Detect/Locate malicious faulty processors/agreement-servers

Each fault-free processor/agreement-server examines all vertices (except vertex $s\ldots\mu$) in the ***IC-trees*** in a top-down and level by level sequence, where μ is Processor/agreement-server id and Processor/agreement-server μ has been detected as an *away processor* or malicious faulty processor/agreement-server**.** If the number of the most common value in vertex $s\ldots\ddot{\imath}$ is less than *threshold-MANET*/ *threshold-CN*, then Processor/agreement-server $\ddot{\imath}$ is a malicious faulty processor/agreement-server. Each processor/agreement-server sets $MFN = MFN \cup \{\ddot{\imath}\}$, where $\ddot{\imath}$ is Processor/agreement-server id. The detail description about *threshold-MANET* and *threshold-CN* is shown in Lemma 7-1-3 and Lemma 7-2-3.

7.1.3 Re-configuration Phase

In this phase, each processor/agreement-server re-configures the network logically by isolating processors/agreement-servers in the set of *ISOLATION*. The set of *ISOLATION*=*MFN*$\cup$(*AN*-*RN*) is used to record processors/agreement-servers that should be isolated. Then, each processor/agreement-server sets *AN*=Null, *RN*=Null and *MFN*=Null.

ADAPTIVE FAULT DIAGNOSIS AGREEMENT PROTOCOL (AFDA)

Message-Collection Phase:

Step1:

For MANET:

Each processor (except *away processor* and *return processor*) uses MAHAP to distribute its ic-tree (as the initial value) to all processors.

For combined wired/wireless network:

Each agreement-server (except *away processor* and *return processor*) uses SBAP to distribute its ic-tree (as the initial value) to all agreement-servers.

Step2:

Then each processor/agreement-server (depends on the network) stores the other processors'/agreement-servers' ic-trees to construct the set of ***IC-trees*** =[ic-tree$_{s,}$ ic-tree$_{a}$, ic-tree$_{b}$ …, ic-tree$_{\ddot{v}}$], where $\ddot{v}$ is the last Processor/agreement-server id in the network by alphabetical order.

=>Each fault-free processor/agreement-server constructs the same set of ***IC-trees*** (the common set of ***IC-trees***) by using MAHAP/SBAP.

Fault-Diagnosis Phase:

- ◆ Set *MFN*=Null; the set of *MFN* is used to record malicious faulty processors/agreement-servers.
- ◆ Set *AN*=Null; the set of *AN* is used to record processors which has ever moved away.
- ◆ Set *RN*=Null; the set of *RN* is used to record *return processors*.

$MFN = MFN \cup$ {malicious faulty processors/agreement-servers}.

$AN = AN \cup$ {*away processors*}.

$RN = RN \cup$ {*return processors*}.

```
Step1: Detect/locate away processors and return processors
    1.1  If Processor $\rho$ is an away processor, Then
             Set $AN = AN \cup \{\rho\}$, where $\rho$ is Processor id.
         End if
    1.2  If Processor Ъ is a return processor, Then
             Set $RN=RN \cup \{Ъ\}$, where Ъ is Processor id.
         End if
```

Figure 7-1. The proposed AFDA protocol *(cont'd.)*

```
Step2: Detect/locate malicious faulty processors/agreement-server
       Parameter $threshold\text{-}MANET = n\text{-}(|AN| + \lfloor (n\text{-}|AN|\text{-}1)/3 \rfloor)$.
       Parameter $threshold\text{-}CN = n\text{-}(\lfloor (n\text{-}1)/3 \rfloor)$.
       Examine all vertices (except vertex $s\ldots\mu$) in the IC-trees by the following rule (in a
       top-down and level by level sequence), where $\mu$ is Processor id and Processor $\mu$ has
       been detected as an away processor or malicious faulty processor/agreement-server.
         For MANET:
           If the number of the most common value in vertex $s\ldots ï$ is less than
           threshold-MANET, Then
              Processor ï is a malicious faulty processor.
              Set $MFN = MFN \cup \{ï\}$, where ï is Processor id.
           End if
         For combined wired/wireless network:
           If the number of the most common value in vertex $s\ldots ï$ is less than
           threshold-CN, Then
              Agreement-server ï is a malicious faulty agreement-server.
              Set $MFN = MFN \cup \{ï\}$, where ï is agreement-server id.
           End if
Re-Configuration Phase:
  Step1: Set $ISOLATION = MFN \cup (AN\text{-}RN)$; The set of ISOLATION is used to record
         processors/agreement-server which should be isolated.
  Step2: According to ISOLATION, each processor/agreement-server can re-configure the
         network logically.
  Step3: Set AN=Nul, RN=Null and MFN=Null.
```

Figure 7-1. The proposed AFDA protocol

7.2 An AFDA Execution Example

The evidence-based FDA protocol AFDA is based on the BA protocol MAHAP/SBAP. An MAHAP execution example is given in chapter 4.3. Hence, we give an example of executing AFDA with MAHAP. That is, AFDA collects all the processors' ic-trees as evidence from the example in chapter 4.3.

7.2.1 Message-Collection Phase

Each processor (except *away processor* and *return processor*) uses MAHAP to distribute its ic-tree from the example in chapter 4.3 to all processors in the message-collection phase. Then, each fault-free processor constructs the same set of ***IC-trees*** =[ic-tree$_s$, ic-tree$_a$, …, ic-tree$_h$] as shown in Figure 7-2.

7.2.2 Fault-Diagnosis Phase

Each processor can detect and locate Processor *b* and Processor *f* that have ever been *away processors*, so it sets $AN = AN \cup \{b, f\}$. Because Processor *b* is also a *return processor*, each processor also sets $RN=RN \cup \{b\}$.

Since AFDA examines vertices in a top-down and level by level sequence, so AFDA examines all values in vertices *s* of the common set of ***IC-trees*** at first. The values stored in vertex *s* of ***IC-trees*** are (0,0,0,0,1,1,1). The number of the most common value “0” stored in the vertex *s* is 4, which is less than 5 (9-(2+2)). Each fault-free processor can detect/locate that Processor *s* is a malicious faulty processor and set $MFN=MFN \cup \{s\}$. Then, each fault-free processor examines the vertices *sa*, *sb*, *sc*, *sd*, *se*, *sf*, *sg*, and *sh* in level 2. Since Processor *b* and Processor *f* have been detected as *away processors*, each processor will not examine the vertices *sb* and *sf*. In level 3, each processor can detect/locate Processor that *e* is a malicious processor by examining vertex *sae*. Because, the values stored in vertex *sae* are (0,1,0,1,0,1,1), the number of the most common value “1” is 4 which is less than 5 (9-(2+2)). Each processor sets $MFN=MFN \cup \{e\}$.

7.2.3 Re-configuration Phase

According to $AN=\{b, f\}$, $RN=\{b\}$ and $MFN=\{s,e\}$, $ISOLATION=\{s,e\} \cup \{b, f\}-\{b\}=\{s,e,f\}$.

Each processor can isolate the Processor *s*, Processor *e* and Processor *f* to re-configure the MANET logically as shown in Figure 7-3. Finally, it sets *AN*=Null, *RN*=Null and *MFN*=Null. Furthermore, due to the fact that the source processor (leader) *s* is a faulty processor, the system should elect a new source processor in the MANET [31] .

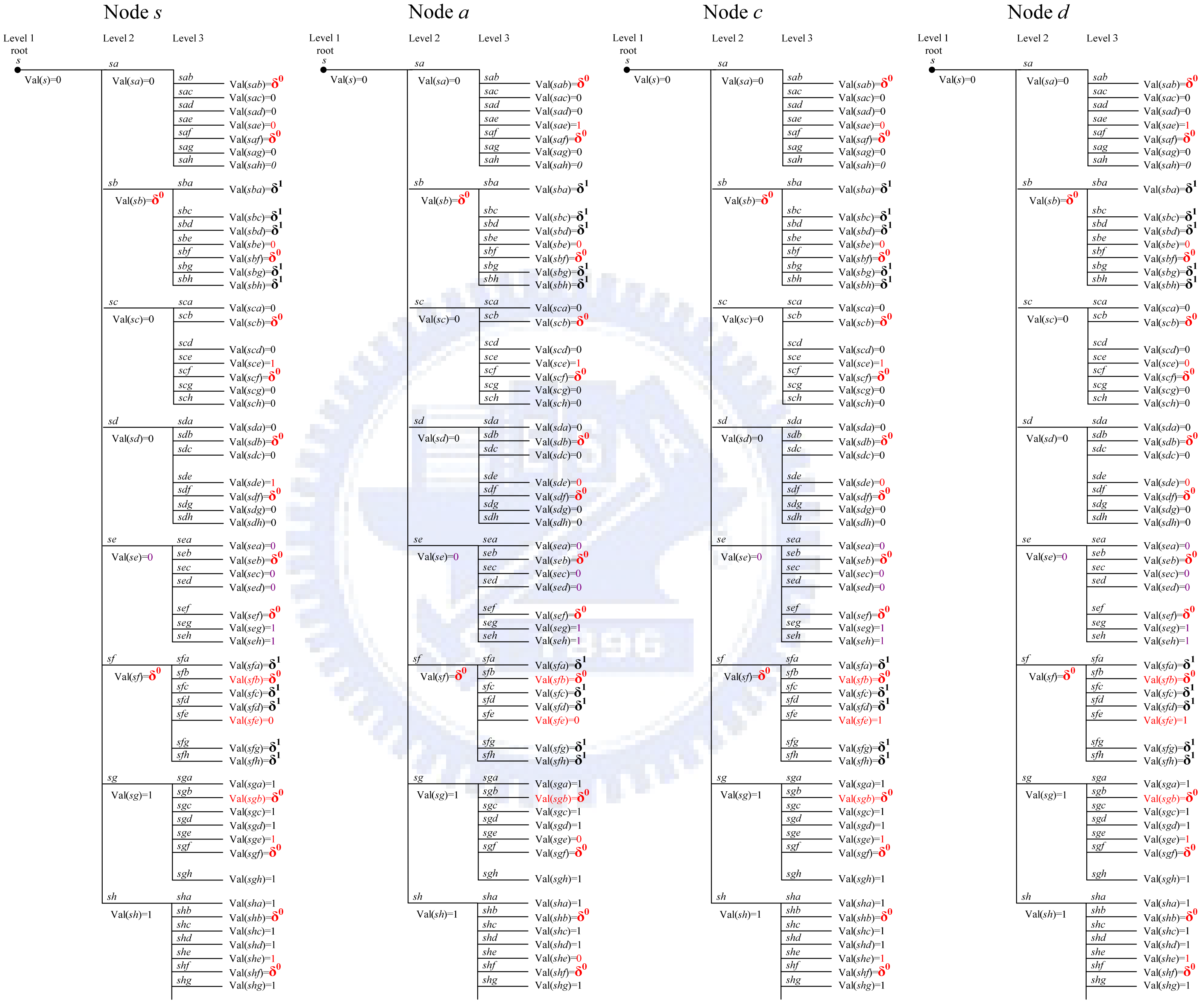


Figure 7-2. The common set of the ***IC-trees*** by each fault-free processor *(cont'd.)*

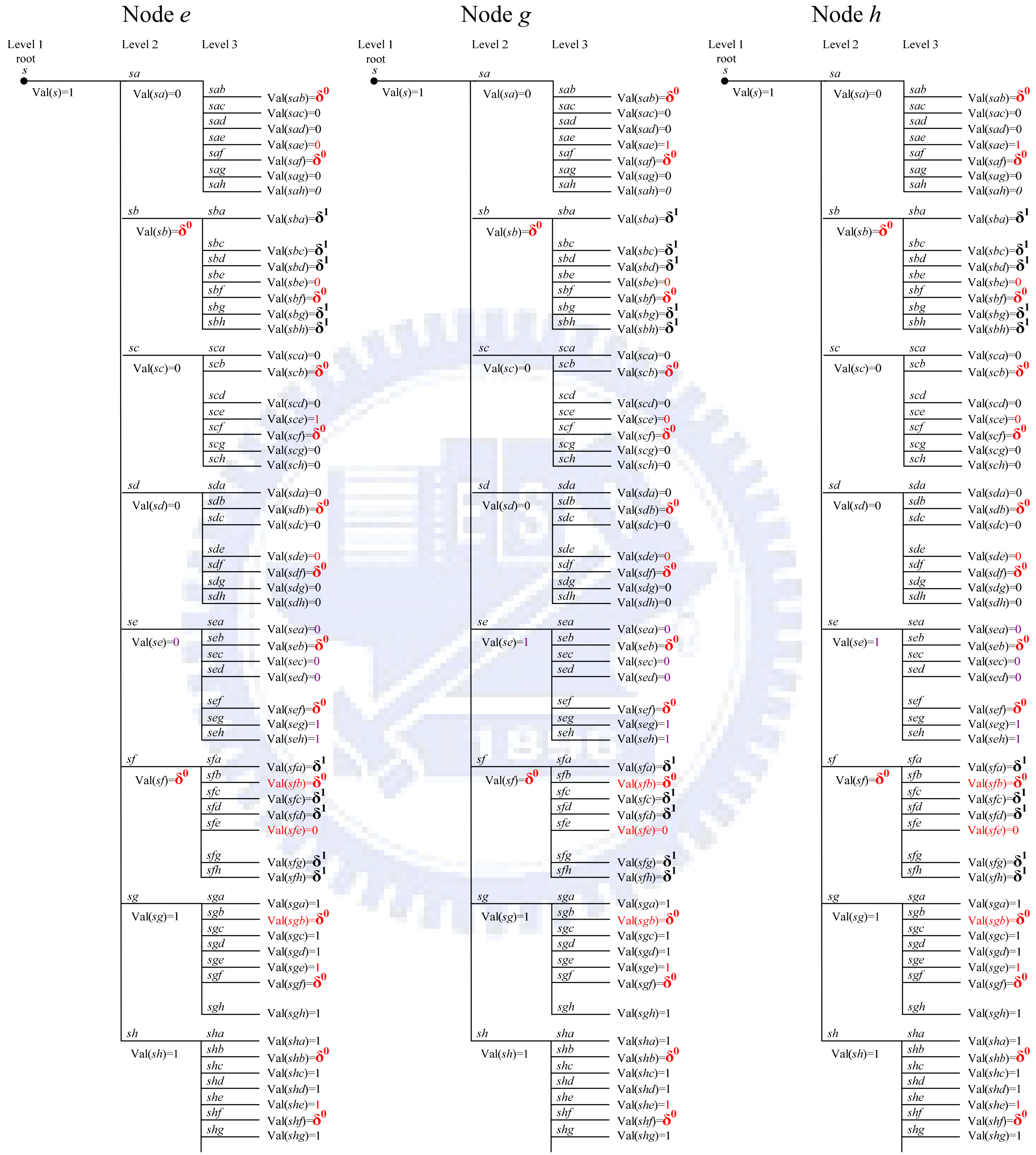


Figure 7-2. The common set of the ***IC-trees*** by each fault-free processor

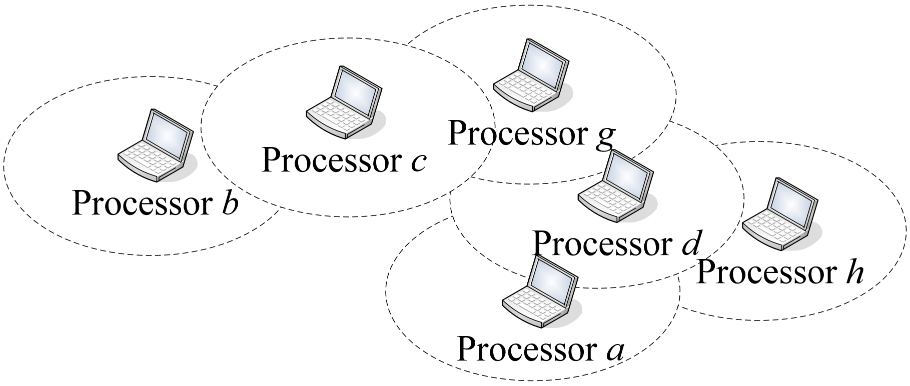


Figure 7-3. An example of MANET after re-configuration

7.3 The Correctness of AFDA

7.3.1 AFDA with MAHAP in Wireless Network

Lemma 7-1-1: Each fault-free processor receives the same common set of *IC-trees* as evidence in the <u>message-collection phase</u> by using AFDA and MAHAP.

Proof: The proposed BA protocol MAHAP satisfies the following requirements in the MANET:

(BA_Agreement): ***All*** fault-free processors agree on a common value;

(BA_Validity): If the source (commander) processor is fault-free, then ***all*** fault-free processors agree on the initial value that the source processor sends.

After executing the BA protocol MAHAP, each fault-free processor reaches the same agreement value whether the source processor is fault-free or not. That is, if the source processor is fault-free, all fault-free processors in the network must reach a common agreement value which is the initial value of the source processor. If the source processor is a faulty processor, all fault-free processor also reach a common agreement value.

In order to let each fault-free processor collect the same set of ***IC-trees*** in the MANET, each processor distributes its ic-tree to all the other processors by using

MAHAP. Finally, each fault-free processor can receive the same set of ***IC-trees***=[ic-tree$_{s,}$ ic-tree$_{a}$, ic-tree$_{b}$ …, ic-tree$_{ϋ}$], where *ϋ* is the last Processor id in the MANET by BA_Agreement and BA_Validity. ■

Lemma 7-1-2: Each fault-free processor can detect/locate the same faulty processors by using AFDA and MAHAP.

Proof: Each fault-free processor receives the same evidence by Lemma 7-1-1 and uses the same FDA protocol AFDA, so each fault-free processor will surely detect/locate the same faulty processors. ■

Theorem 7-1-1: Protocol AFDA with MAHAP satisfies the agreement of FDA.

Proof: By Lemma 7-1-1 and Lemma 7-1-2, AFDA can identify the common set of faulty processors. ■

Lemma 7-1-3: The malicious faulty processors can be detected and located in MANET if $n>3p_m+p_a$.

Proof: According to the constraint on the number of processors in an MANET is $n>3p_m+p_{a,}$ there are at most p_a *away processors* and p_m malicious processors in an MANET. So, p_a ($p_a=|AN|$) *away processors* cannot transmit message and p_m malicious processors can produce at most p_m values at the same vertex in the ***IC-trees*** different from the most common value. That is, if the sender Processor *ï* is fault-free, there are at least n-($|AN|+\lfloor(n-|AN|-1)/3\rfloor$) values at the vertex *s…ï* in the ***IC-trees*** are the same. Otherwise, the Processor *ï* is a malicious faulty processor.

Lemma 7-1-4: The maximum number of detectable/locatable faulty processors and *away processors* by AFDA in MANET is p_m malicious faulty processors and p_a away processors, $n>3p_m+p_a$.

Proof: Due to the constraint on the number of processors in an MANET is $n>3p_m+p_{a,}$ there are at most p_m malicious processors and p_a *away processors* in an MANET. ■

Theorem 7-1-2: Protocol AFDA with MAHAP satisfies the fairness requirement of FDA.

Proof: By Lemma 7-1-3, no fault-free processor is falsely detected as faulty by any fault-free processors if $n>3p_m+p_a$. ■

Theorem 7-1-3: Protocol AFDA with MAHAP solves the FDA problem in an MANET if $n>3p_m+p_a$.

Proof: By Theorem 7-1-1 and Theorem 7-1-2, this theorem is proved. ■

7.3.2 AFDA with SBAP in Combined Wired/Wireless Network

Lemma 7-2-1: Each fault-free agreement-server receives the same common set of *IC-trees* as evidence in the message-collection phase by AFDA and SBAP.

Proof: The proposed BA protocol SBAP satisfies the following requirements in the combined wired/wireless network:

(BA_Agreement): All fault-free processors managed by fault-free agreement-server agree on a common value;

(BA_Validity): If the source (commander) agreement-server is fault-free, the agreement value should be the initial value of the source agreement-server.

After executing the BA protocol SBAP, all fault-free processors managed by fault-free agreement-server agree on a common value whether the source agreement-server is fault-free or not. That is, if the source agreement-server is fault-free, all fault-free processors managed by fault-free agreement-server in the network must reach a common agreement value which is the initial value of the source agreement-server. If the source agreement-server is a faulty processor, all fault-free processors managed by fault-free agreement-server also reach a common agreement value.

In order to let each fault-free processor managed by fault-free agreement server collect the same set of ***IC-trees*** in the combined wired/wireless network, each agreement-server distributes its ic-tree to all the other agreement-server by using SBAP. Finally, each fault-free agreement-server can receive the same set of ***IC-trees***=[ic-tree$_{s,}$ ic-tree$_a$, ic-tree$_b$ …, ic-tree$_{\ddot{v}}$], where $\ddot{v}$ is the last agreement-server id in the combined wired/wireless network by BA_Agreement and BA_Validity. ■

Lemma 7-2-2: Each fault-free agreement-server can detect/locate the same faulty agreement-server by using AFDA and SBAP.

Proof: Each fault-free agreement-server receives the same evidence by Lemma 7-2-1 and uses the same FDA protocol AFDA, so each fault-free agreement-server will surely detect/locate the same faulty agreement-servers. ■

Theorem 7-2-1: Protocol AFDA with SBAP satisfies the agreement of FDA.

Proof: By Lemma 7-2-1 and Lemma 7-2-2, AFDA can identify the common set of faulty agreement-servers. ■

Lemma 7-2-3: The malicious faulty agreement-servers can be detected and located in combined wired/wireless network if $z_n>3z_{m.}$

Proof: According to the constraint on the number of agreement-server in a combined wired/wireless network is $z_n>3z_{m,}$ there are at most z_m malicious agreement-servers in a combined wired/wireless network. So, z_m malicious processors can produce at most z_m values at the same vertex in the ***IC-trees*** different from the most common value. That is, if the sender agreement-server $\ddot{\imath}$ is fault-free, there are at least n-($\lfloor(n$-1)/3$\rfloor$) values at the vertex $s\ldots\ddot{\imath}$ in the ***IC-trees*** are the same. Otherwise, the agreement-server $\ddot{\imath}$ is a malicious faulty agreement-server.

Lemma 7-2-4: The maximum number of detectable/locatable malicious faulty agreement-servers by AFDA in combined wired/wireless network is z_m malicious faulty agreement servers, $z_n>3z_m$.

Proof: Due to the constraint on the number of agreement-servers in a combined wired/wireless network is $z_n>3z_{m,}$ there are at most z_m malicious agreement-servers in a combined wired/wireless network. ■

Theorem 7-2-2: Protocol AFDA with SBAP satisfies the fairness requirement of FDA.

Proof: By Lemma 7-2-3, no fault-free agreement-server is falsely detected as faulty by any fault-free agreement-server if $z_n>3z_m$. ■

Theorem 7-2-3: Protocol AFDA with SBAP solves the FDA problem in a combined wired/wireless network if $z_n>3z_m$.

Proof: By Theorem 7-2-1 and Theorem 7-2-2, this theorem is proved. ■

7.4 Conclusion

In order to provide a highly reliable computing environment for combined wired/wireless network, we need to solve the FDA problems in combined wired/wireless network. In previous result, FDA problems were considered for a static network. Therefore, in this study, we revisited the FDA problems in the combined wired/wireless network. AFDA is an adaptive FDA protocol. AFDA not only can solve the FDA problem in combined wired/wireless network, but also AFDA can solve the FDA problem in other networks. The proposed evidence-based FDA protocol AFDA with MAHAP can detect/locate at most p_m malicious faulty processors in an MANET. AFDA with SBAP can detect/locate at most z_m malicious faulty agreement-servers in a combined wired/wireless network.

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

Chapter 8

Consensus Problem under Peer-to-Peer Environment: An Application to File-Sharing

In nowadays, Peer-to-Peer networks (P2P) have been more and more popular. Many P2P networks are overlay networks because they run on top of the combined wired/wireless network. We know that the most popular application in Peer-to-Peer (P2P) system is file-sharing. However, malicious attackers may modify files arbitrarily and spread inconsistent files to other processors in the P2P network; inconsistent files will not only spread in P2P networks but also waste resources, such as bandwidth, space of storage and transmission time. Hence, how to make fault-free processors ensure that the files they hold are correct is an import topic. So far, no previous study has attempted to solve the Consensus problem of file-sharing with malicious processors in P2P networks.

In this chapter, we give an application of Consensus protocol. We proposed a novel hybrid approach to solve the Consensus problem of file-sharing in P2P systems. Moreover, a P2P network is composed of heterogeneous processors, and the ability of each processor may vary with its computation capability (CPU), bandwidth, space of storage and etc. To provide better quality of service (QoS), we should group processors by their abilities to reduce waiting time. In this study, the clustering algorithm is employed to cluster the large sets of processors into groups of smaller sets of similar processors. That is, we propose a novel hybrid approach that clusters similar processors into the same group to provide better QoS and solve the Consensus problem of file-sharing with malicious processors in P2P networks.

8.1 Introduction

This section introduces the classification of peer-to-peer file-sharing systems and the clustering algorithm.

8.1.1 The Classification of Peer-to-Peer File-Sharing Systems

Peer-to-Peer (P2P) systems have been developed for a long time and become more and more popular in the area of file-sharing [2] [11] [17] [34] [20] [29] [35] [42] [49] [54] . P2P file-sharing systems can be classified into two categories, centralized P2P file-sharing and distributed P2P file-sharing. An example of centralized P2P file-sharing system is Napster [34] which was first introduced in January 1999 by Shawn Fanning for MP3 files sharing and later unplugged in July 2001. In a centralized system, each client can look up files through centralized servers effectively. However, the centralized P2P file-sharing system has the problem of “single point of failure”. Contrary to centralized P2P file-sharing, distributed P2P file-sharing systems do not have this problem. The distributed P2P file-sharing systems can be subdivided into structured and unstructured systems.

In the unstructured P2P file-sharing system, such as Gnutella [20] , each node incurs a reasonable overhead to build overlay links arbitrarily. Lookup service in the Gnutella is performed using query flooding. However, using query flooding in an unstructured P2P file-sharing system may not accurately locate the file that is really in the system. Because the network is unstructured and query packet is forwarded to the system until its Time-To-Live (TTL) becomes zero.

In a structured P2P file-sharing system, both overlay topology and file placement are tightly controlled. The most common structured P2P file-sharing system is Distributed Hash Table (DHT) system [29] [35] [42] [49] [53] [54] . Distributed hash tables are a class of distributed systems that provide lookup services similar to a hash table: (name, value) pairs are stored in the DHT. Any participating processor can efficiently retrieve the value associated with a given name [18] . The structured DHT P2P file-sharing system can also be further classified into two categories: variable-degree DHT system (e.g., CAN [42] and Chord [54]) and constant-degree DHT system (e.g., de-Bruijn [29] [53]). A constant-degree DHT system, such as de-Bruijn, has a constant-sized routing table and achieves a log-arithmetic routing hops. Moreover, de-Bruijn graphs are nearly optimal [29] . A detailed description of de-Bruijn will be provided in chapter 8.2.3. In this study, we adopt the de-Bruijn graph as overlay network in our P2P system. The classification of P2P file-sharing systems is shown in Figure 8-1.

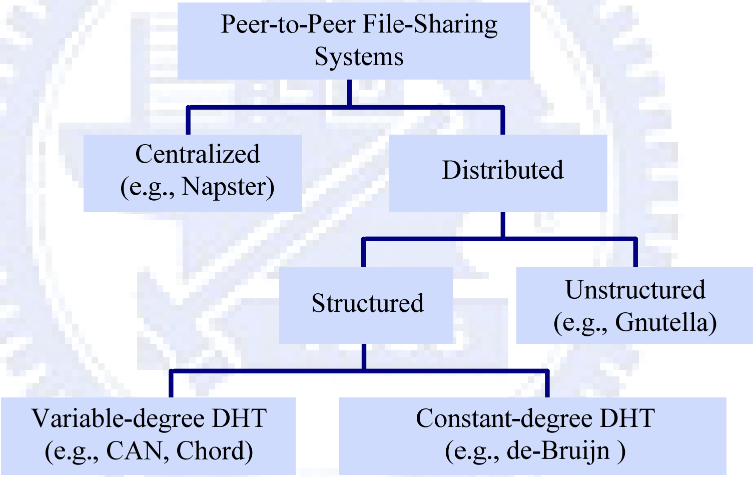

Peer-to-Peer File-Sharing Systems
Centralized (e.g., Napster)
Distributed
Structured
Unstructured (e.g., Gnutella)
Variable-degree DHT (e.g., CAN, Chord)
Constant-degree DHT (e.g., de-Bruijn)

Figure 8-1. Classification of P2P file-sharing systems

8.1.2 Clustering Algorithm

P2P network is composed of a large number of heterogeneous processors. The ability of each processor in P2P network may vary with several factors, including computation capability (CPU), bandwidth, space of storage, and etc [43] . However, putting varied processors in the same group may reduce system performance. Higher-performance processors may spend a lot of time waiting for lower-performance processors. In order to provide better Quality of

Service (QoS), we should put processors with similar abilities into the same group to reduce the waiting time. Through grouping, processors within the same group are more similar to each other than those in other groups [69] . Hence, we introduce the clustering algorithm, k-means algorithm [26] , to find natural groups of data (processors) based on computation capability, bandwidth, and space of storage. A detailed description of k-means algorithm is provided in chapter 8.2.1.

8.2 System Model and Approach

In this section, we introduce our approach to solve the Consensus problem of file-sharing with malicious processors in P2P systems. Figure 8-2. shows the flow chart of our approach.

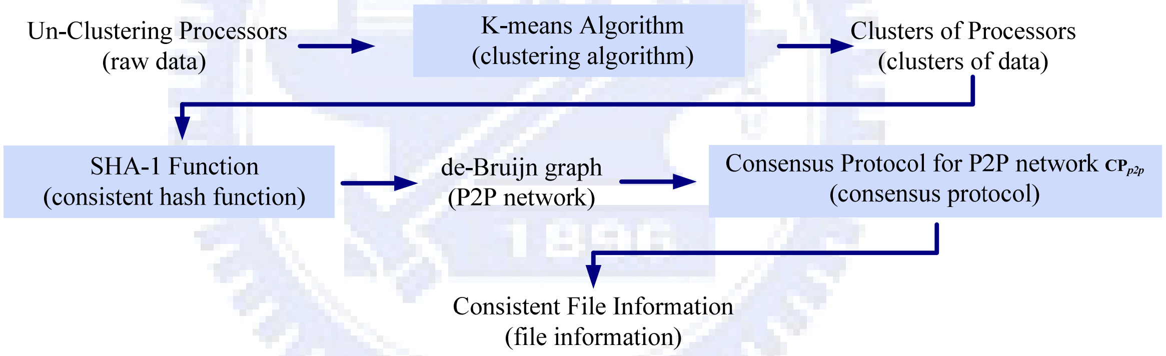


Figure 8-2. The flow chart of the proposed approach

8.2.1 Clustering Algorithm: K-means Algorithm

The most important factors to consider when evaluating the ability of a processor are computation capability, bandwidth, and space of storage [43] . In order to enhance system efficiency, we should cluster processors with similar abilities into the same group based on computation capability, bandwidth, and space of storage to reduce the waiting time. The k-means algorithm is the simplest and most popular clustering algorithm [26] . Hence, we adopt k-means algorithm as our clustering algorithm in our approach.

The k-means algorithm is performed in the following procedure: Suppose there are n_t un-clustered processors described by the attribute vectors S={x_1, x_2,⋯, x_{n_t} }. We want to partition these n_t processors in to k clusters, where $k< n_t$, $S = S_1 \cup S_2 \cup \ldots \cup S_k$. Let c_i be the mean of the vectors in cluster i (S_i). First of all, select k number of points to be the centers of clusters arbitrarily, C={c_1, c_2,⋯, c_k}, and then find the Euclidean distance from each point to each center, and assign the point to the closest center. Next, for each set of points assigned to a center, find the middle of the cluster, take that value as the new center, and repeat the process until all the centers are fixed. The k-means algorithm is shown in Figure 8-3.

- n_t: the total number of un-clustered processors.
- k: the number of clusters, $k< n_t$.
- S: the set of instances, S={x_1, x_2,⋯, x_n}, $|S|= n_t$, $S = S_1 \cup S_2 \cup \ldots \cup S_k$.
- C: the set of centers in each cluster, C={c_1, c_2,⋯, c_k}.
- $S_i(t)$: the set S_i in the t-th iteration.
- N_i: the number of processors in the set $S_i(t)$.
- $c_i(t)$: the center of S_i in the t-th iteration.
- $||x_m-c_i(t)||$: the Euclidean distance between x_m and c_i.
- Random_Select(k,S): select k processors from the set S randomly, C={c_1, c_2,⋯, c_k};

```
Input:
    Number of clusters, $k$;
    The set of instances, $S=\{x_1, x_2,\cdots, x_{n_t}\}$;
/* Initialization */
    $C$ = Random_Select($k$,$S$);          /* $C=\{c_1, c_2,\cdots, c_k\}$ */
/* Partition */
    Repeat
    $x_m \in S_j(t), if \|x_m - c_j(t)\| < \|x_m - c_i(t)\|, m = 1,\ldots,n_t; i, j = 1,\ldots,k, i \neq j$;
    $N_j = |S_j(t)|$;
    $c_j(t+1) = \frac{1}{N_j}\sum_{x \in S_j(t)} x, j = 1,\ldots,k$;
    Until $c_j(t+1) = c_j(t)$
Output:
     Set $S_1, S_{2,\ldots,} S_k$;
```

Figure 8-3. The k-means algorithm

8.2.2 Consistent Hash Function: SHA-1 Function

After clustering, we use SHA-1 function [18] to assign each processor and key to the corresponding processor in the de-Bruijn network, A detailed description of de-Bruijn network will be provided in chapter 8.2.3. The SHA-1 function is a consistent hash function that assigns each processor and key an *m*-bit *identifier*. A processor's identifier is defined by hashing the processor's IP address, while a key identifier is produced by hashing the key.

- Processor's ID = SHA-1 (IP address)
- Key's ID = SHA-1 (object's key/name)

8.2.3 Overlay Network: de-Bruijn Graph

The performance of P2P network is determined by the properties of diameter and degree of graph. Since non-trivial Moore graphs are non-existent [7] , Loguinov et al. [29] indicated that de-Bruijn graphs of diameter $\lceil \log_k n \rceil$ are nearly optimal and feature very short average routing distances and high resilience to processor failure. Hence, we adopt de-Bruijn graph as overlay network in our system model.

The de-Bruijn graph is denoted by DB(h,d), where $h \geq 2$ is the *node degree* and d is the *dimension* of the graph. The de-Bruijn graph has $n=h^d$ processors and processor is encoded by a h-ary string of d digits from the digit set {0, 1, 2, …, h-1}. The DB graphs are *directed* graphs with h incoming edges and h outgoing edges. An example of de-Bruijn graph of degree 2 and dimension 3 is shown in Figure 8-4.

Shortest path routing in de-Bruijn graphs is easy to implement by the string-matching algorithm [29] . Assume that processor x wants to seek a shortest path to processor y. Then processor x finds the longest *overlap* between the *suffix* of its id and *prefix* of y's id as shown in Figure 8-5. For example, processor 000 wants to find the shortest path to processor 011. According to the string-matching algorithm in Figure 8-5, the prefix A is 00, overlap B is 0, and suffix C is 11. Hence, the routing path P is 00011, and the shortest routing path is 000=>001=>011.

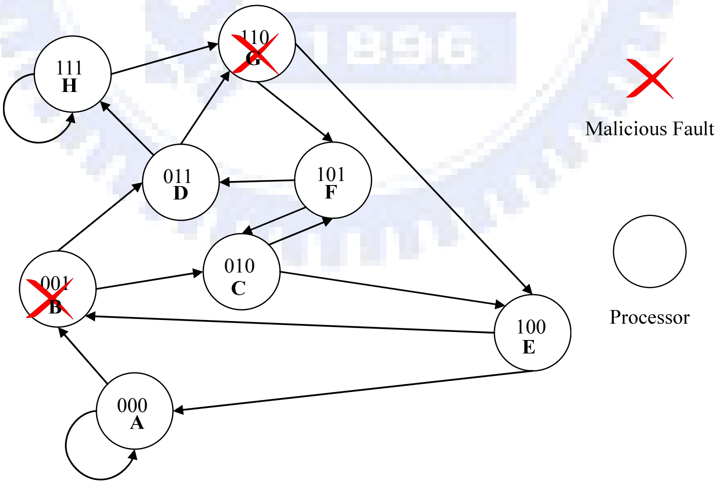


Figure 8-4. An example of DB(2,3) de-Bruijn network

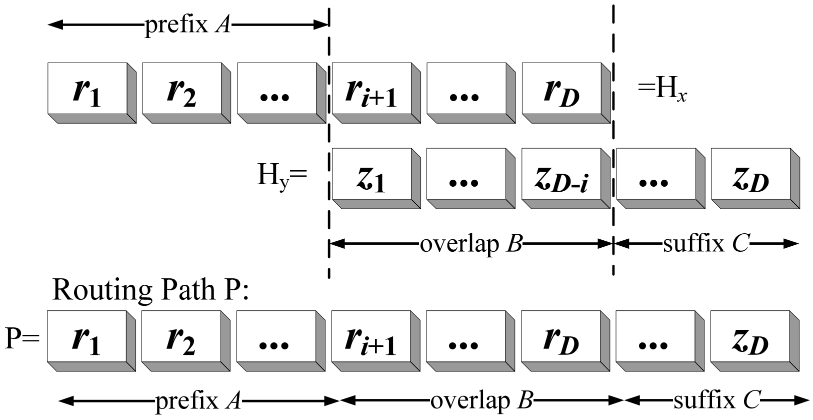


Figure 8-5. The shortest path from H_x to H_y

8.2.4 Consensus Protocol: Consensus Protocol for P2P Network (CP$_{p2p}$)

Previous studies on P2P fault tolerance mostly dealt with random attacks [2] [17] [29] [49] , by which each peer suffers a fault independently. However, these solutions cannot handle processor collusions effectively. In order to get the consistent file information from fault-free processors, we propose the "Consensus Protocol for P2P Network" (CP$_{p2p}$) to solve the Consensus problem under malicious attacks from processor collusion in the de-Bruijn P2P file-sharing system.

The goal of CP$_{p2p}$ is to get the consistent file information from fault-free processors. The file information contains length, name, size, and etc [11] . There are two phases in the protocol CP$_{p2p}$: message-exchanging phase and decision-making phase. In the message-exchanging phase, we use the term "round" to compute the number of messages exchanged. A round is defined as follows: (i) Sends messages to other processors (ii) Receives messages from this round (iii) Does local processing [19] [60] [70] Fischer and Lynch [19] indicated that t+1 (t=$\lfloor(n\text{-}1)/3\rfloor$) rounds are the minimum number of rounds required to get enough messages to achieve agreement if and only if the maximum number of malicious processors is smaller than 1/3 of the total number of processors, where n is the total number of processors in the network. The number of rounds required in protocol CP$_{p2p}$ is also t+1, which is the minimum. The decision-making phase is used to compute the Consensus value. The procedure of removing influence from malicious processors (including malicious intermediate processors and malicious sender processors) by CP$_{p2p}$ is shown in Figure 8-6.

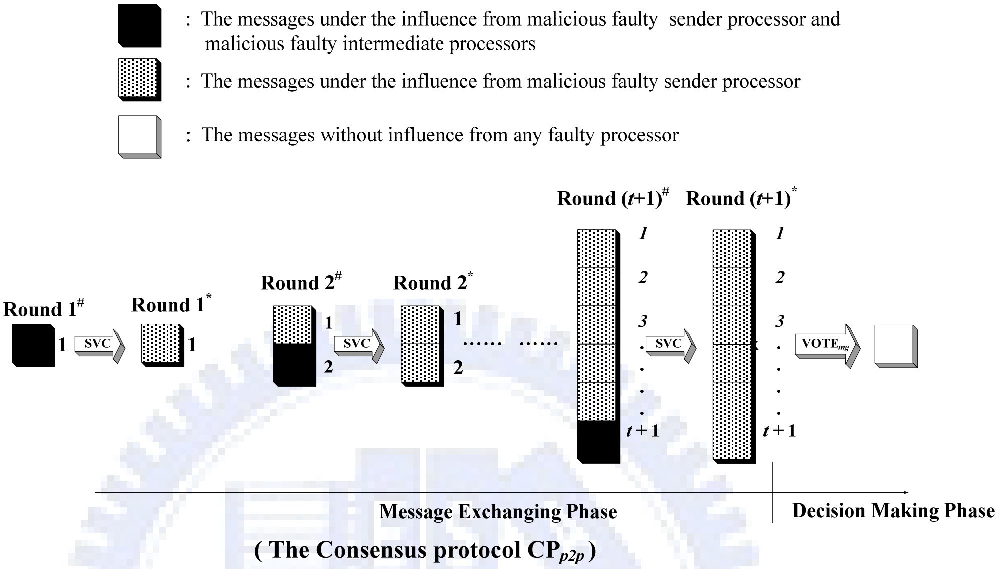

: The messages under the influence from malicious faulty sender processor and malicious faulty intermediate processors
: The messages under the influence from malicious faulty sender processor
: The messages without influence from any faulty processor
Round 1#
Round 1*
Round 2#
Round 2*
Round (t+1)#
Round (t+1)*
SVC
VOTEmg
Message Exchanging Phase
Decision Making Phase
(The Consensus protocol CPp2p)

Figure 8-6. The procedure of removing influence from maliciously colliding processors

CP$_{p2p}$ combines the following approaches to solve the Consensus problem with maliciously colliding processors in de_Bruijn P2P file-sharing system.

- **Secure Communication Channel:** we combine the symmetric cryptographic and asymmetric cryptographic algorithms to take the advantages of both algorithms. There are two phases in the secure communication channel, including the channel-creating phase and the message-transmission phase. In the channel-creating phase, the session key is generated using the Diffie-Hellman key exchange [16] (asymmetric cryptographic algorithm) and the symmetric key is generated using the Advanced Encryption Standard (AES) algorithm [14] (symmetric cryptographic algorithm). The goal of session key is used to distribute the symmetric key to the receiver securely. In the message-transmission phase, we use a symmetric key to encrypt and decrypt the messages to reduce the computation time.

At the beginning of the *i*-th round of message exchange, say Round $i^{\#}$, each sender uses symmetric key to send its messages to all other processors in the network. Hence, each receiver collects all of the senders' messages without being influenced from malicious intermediate processors when symmetric key is applied, say Round i^{*}, as shown in Figure 8-6.

- **The Voting Function – VOTE$_{mg}$:** After t+1 rounds of message exchange, each fault-free processor can receive enough messages to remove the influence from malicious sender processor using the VOTE$_{mg}$ function in the decision-making phase. The VOTE$_{mg}$ function is shown in Figure 8-7.

```
                    Begin
                       If $\alpha$ is a leaf Then
                          output the value of $\alpha$
VOTE$_{mg}$( $\alpha$ ) =     If the majority value is $m$, $m \in \{0,1\}$
                          output the majority value $m$
                       If the majority value does not exist
                          output the default value $\phi$
                    End
```

Figure 8-7. The VOTE$_{mg}$ Function

CP$_{p2p}$ combines secure communication channel and the VOTE$_{mg}$ function to solve the Consensus problem in de-Bruijn overlay P2P file-sharing system. The procedure can be presented with the following primitives:

- SMA(*Þ*): search the shortest path to processor *Þ* by the string-matching algorithm.
- SVC_Send(*m*,*Þ*): send message *m* to processor *Þ* using the symmetric key from protocol SVC.
- SVC_Receive(*m*,*Þ*): receive message *m* from processor *Þ* using the symmetric key

from protocol SVC.

- Create_Vertex(*αÞ*): create vertex *αÞ* in the mg-tree, where *α* is the vertex name (a sequence of processor id) of mg-tree in level *i*-1, |*αÞ*|=*i*, 2≤|*αÞ*|≤*t*+1.
- val(*αÞ*): the value of vertex *αÞ* in the mg-tree.
- Store_Vertex(*m*,*αÞ*): store the message *m* from processor *Þ* in the vertex *αÞ* and set val(*αÞ*)=*m*.
- Get_Vertex(*m*, *i*): get the message *m* from level *i* of the mg-tree.
- Del_Repeated(*mg-tree*): delete vertices with repeated name in the mg-tree.

Using the above primitives, the formal procedure of CP$_{p2p}$ is shown in Figure 8-8.

```
/* Initialization */
    Create_Vertex(R);
/* Message-Exchanging Phase */
/* the first round */
    for Þ∈N do
        SMA(Þ);
        SVC_Send(m,Þ);
    end
    for Þ∈N do
        SVC_Receive(m,Þ);
        Create_Vertex(Þ);
        Store_Vertex(m,Þ);
    end
/* round i, 2≤i≤t+1 */
    for Þ∈N do
        Get_Vertex(m, i-1);
        SVC_Send(m,Þ);
    end
    for Þ∈N do
        SVC_Receive(m,Þ);
        Create_Vertex(αÞ);
        Store_Vertex(m, αÞ);
    end
/* Decision-Making Phase */
    Del_Repeated(mg-tree);
    Output(VOTE_mg(R));
```

Figure 8-8. The Consensus Protocol for P2P network (CP$_{p2p}$)

After each processor using CP$_{p2p}$ to send its file information to all other processors, each fault-free processor can get the consistent file information if and only if the maximum number of malicious processors is smaller than 1/3 of the total number of processors in de-Bruijn overlay P2P file-sharing system.

8.3 An CP$_{p2p}$ Execution Example

In this section, we use an example to demonstrate our proposed approaches.

8.3.1 Clustering

As mentioned above, the most important factors considered in evaluating the ability of a processor are computation capability, bandwidth, and space of storage. To simulate the heterogeneity of processors in a P2P file-sharing system, the computation capability, bandwidth, and space of storage of each processor were generated at random. Figure 8-9(a) shows an example of 30 processors generated at random. Then, the k-means algorithm, as shown in Figure 8-3, is applied to group the 30 processors into three clusters. The result of the three clusters is shown in Figure 8-9(b).

8.3.2 Mapping the Processors in Cluster to de-Bruijn Overlay Network

We use the cluster 1 as our example to show how the consistent file information is obtained. First, we map the processors in the cluster 1 to the corresponding processors in de-Bruijn overlay network. Since there are eight processors in the cluster 1, we need to produce a de-Bruijn network with at least eight processors (DB(2,3)). Later, the hash function SHA-1 is used to map the processors in cluster 1 to the corresponding processors in DB(2,3) de-Bruijn network as shown in Figure 8-4. For simplify our descriptions, we use A to represent processor 000, B to represent processor 001, and so on. There are two malicious processors in the DB(2,3) de-Bruijn network; they are processor B and processor G. Hence, processor B and processor G may work in coordination to prevent other fault-free processors from getting the consistent file information and spread the inconsistent files to other fault-free processors.

8.3.3 Getting the Consistent File Information

To prevent the distribution of inconsistent files, each processor executes the protocol CP_{p2p} to get the file information from other processors. Since $\lfloor n\text{-}1/3 \rfloor$ processors at most may be faulty, each fault-free processor can get the correct and consistent file information. The initial value of each processor is shown in Table 8-1.

Table 8-1 The initial value of each processor

Processor ID	A	B	C	D	E	F	G	H
Initial value	1	1	0	0	0	0	1	0

In the beginning of the protocol CP_{p2p} , each processor creates an mg-tree with the vertex $\mathfrak{R}$ in the level 0.

■ **Message-Exchanging Phase**

In the message-exchanging phase, the number of rounds required γ must be computed first, where γ=t+1=3, and t = $\lfloor (n\text{-}1)/3 \rfloor$. In our example, there are eight processors in the DB(2,3) de-Bruijn network as shown in Figure 8-4. Hence, the number of rounds required γ is 3 (γ=$\lfloor (8\text{-}1)/3 \rfloor$+1=3).

In the first round of message-exchanging phase, each processor transmits its initial value to all other processors by protocol SVC. Then each processor stores the messages from other processors in the level 1 of its mg-tree. Figure 8-9(c) shows the mg-trees of each processor after the first round of message-exchanging phase. Since processor B and processor G are malicious processors, they may transmit values arbitrarily.

In the second round of message-exchanging phase, each processor transmits the messages received in the level 1 of its mg-tree to other processors and receives other processors' messages in the level 2 of its mg-tree. An example of processor A's mg-tree after the second round of message-exchanging phase is shown in Figure 8-9(d).

In the third round of message-exchanging phase, each processor transmits the messages received in the level 2 of its mg-tree to other processors and receives other processors' messages in the level 3 of its mg-tree. An example of processor A's mg-tree after the third round of message-exchanging phase is shown in Figure 8-9(e).

■ Decision-Making Phase

In the decision-making phase, each processor deletes vertices with repeated names of mg-tree to avoid the repeated influence from malicious processors. An example of processor A's mg-tree without repeated name vertices is shown in Figure 8-9(f).

After deleting the vertices with repeated names, each processor uses the VOTE_{mg} function on its mg-tree from leaf to root to compute the Consensus value. $\mathrm{VOTE}_{mg}(\Re)=\mathrm{VOTE}_{mg}(\mathrm{VOTE}_{mg}(A), \mathrm{VOTE}_{mg}(B), \mathrm{VOTE}_{mg}(C), \mathrm{VOTE}_{mg}(D), \mathrm{VOTE}_{mg}(E), \mathrm{VOTE}_{mg}(F), \mathrm{VOTE}_{mg}(G), \mathrm{VOTE}_{mg}(H))$. For example, processor A computes $\mathrm{VOTE}_{mg}(\Re)= \mathrm{VOTE}_{mg}(1,1,0,0,0,0,1,0) = 0$, where $\mathrm{VOTE}_{mg}(A)= \mathrm{VOTE}_{mg}(1,1,1,1,1,0,1)$, $\mathrm{VOTE}_{mg}(B)= \mathrm{VOTE}_{mg}(1,1,0,1,0,1,0)$ and so on.

Each processor can find out the inconsistent file by correct file information obtained using protocol CP_{p2p}. Then the fault-free processors know which files they hold are correct and which or not, and they will not share inconsistent files to other processors.

Processor Parameters

ID	Computation capability	Bandwidth	Space of Storage
01	8	10	9
02	7	5	3
03	7	6	5
04	4	9	3
05	7	5	5
06	6	9	7
07	8	9	9
08	1	5	3
09	1	2	1
10	3	9	5
11	6	6	4
12	2	1	1
13	6	4	2
14	1	2	1
15	2	7	5
16	8	6	10
17	5	3	7
18	5	4	3
19	7	9	7
20	7	9	8
21	4	3	4
22	3	7	6
23	7	4	2
24	6	7	8
25	2	7	1
26	6	5	4
27	3	3	1
28	6	4	7
29	10	8	9
30	3	10	2

(a) 30 random processors

Processor Parameters (Cluster 1)

ID	Computation capability	Bandwidth	Space of Storage
01	8	10	9
06	6	9	7
07	8	9	9
16	8	6	10
19	7	9	7
20	7	9	8
24	6	7	8
29	10	8	9

Processor Parameters (Cluster2)

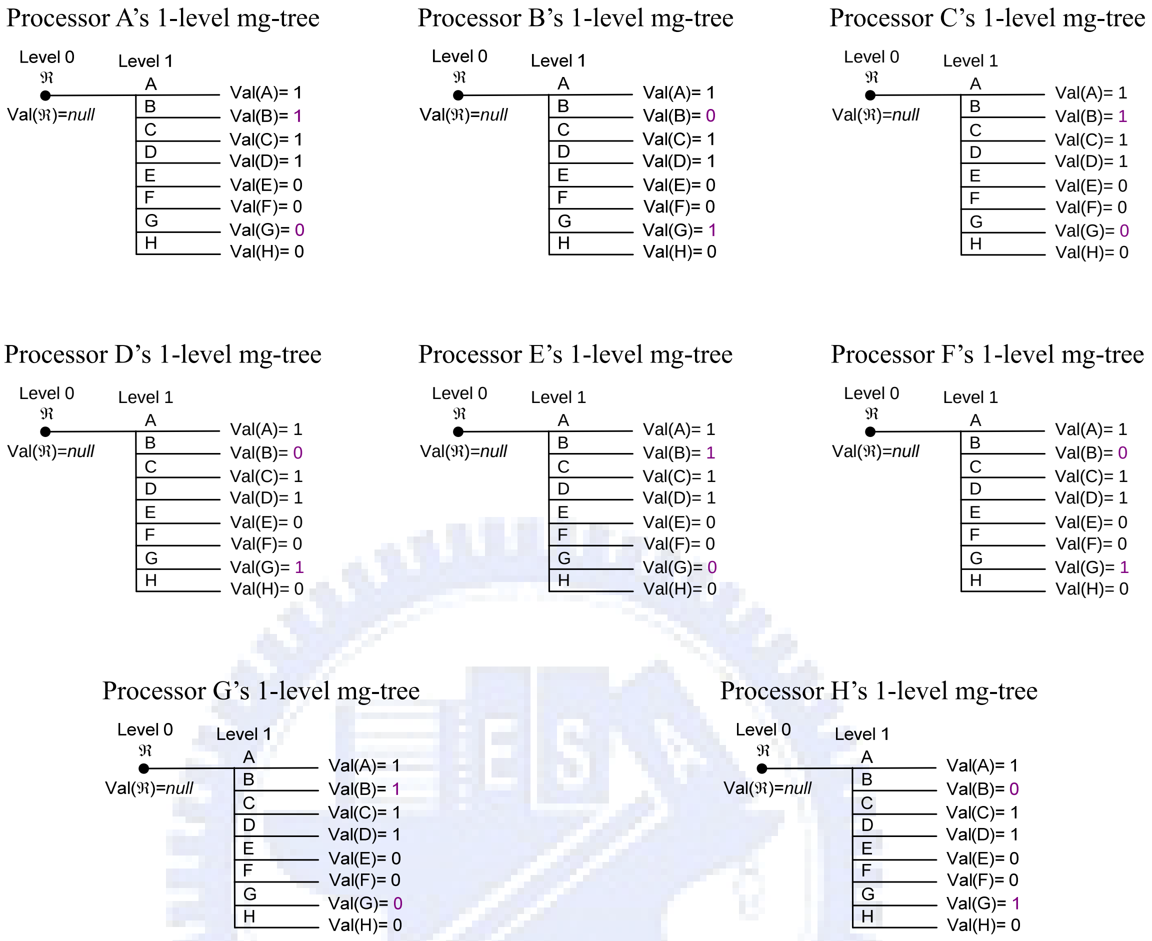
ID	Computation capability	Bandwidth	Space of Storage
02	7	5	3
03	7	6	5
04	4	9	3
05	7	5	5
10	3	9	5
11	6	6	4
13	6	4	2
15	2	7	5
17	5	3	7
18	5	4	3
21	4	3	4
22	3	7	6
23	7	4	2
26	6	5	4
28	6	4	7
30	3	10	2

Processor Parameters (Cluster3)

ID	Computation capability	Bandwidth	Space of Storage
08	1	5	3
09	1	2	1
12	2	1	1
14	1	2	1
25	2	7	1
27	3	3	1

(b) three clusters which are partitioned by k-means algorithm

Figure 8-9. An example of our approach *(cont'd.)*



Processor A’s 1-level mg-tree

Processor B’s 1-level mg-tree

Processor C’s 1-level mg-tree

Processor D’s 1-level mg-tree

Processor E’s 1-level mg-tree

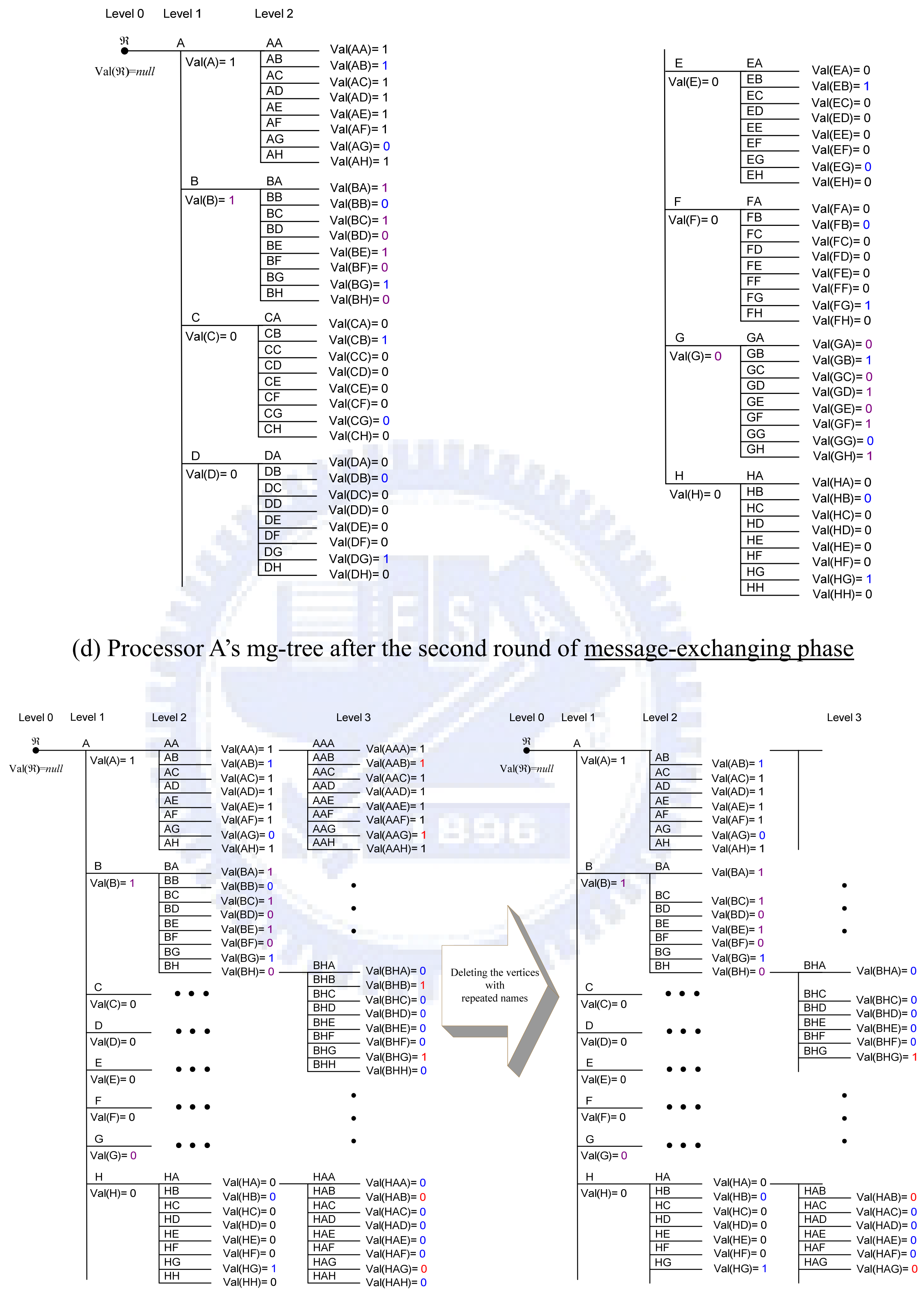
Processor F’s 1-level mg-tree

Processor G’s 1-level mg-tree

Processor H’s 1-level mg-tree

(c) The mg-trees of each processor after the first round of message-exchanging phase

Figure 8-9. An example of our approach *(cont’d.)*



(d) Processor A's mg-tree after the second round of message-exchanging phase

(e) Processor A's mg-tree after the third round of message-exchanging phase

(f) Processor A's mg-tree without repeated name vertices

Figure 8-9. An example of our approach

8.4 The Correctness of CP$_{p2p}$

The following lemmas and theorems are used to prove the correctness of protocol CP$_{p2p}$.

Lemma 8-1: After VOTE$_{mg}$ function is applied to mg-tree from leaf to root, all correct vertices of an mg-tree are common.

Proof: In the decision-making phase, all vertices with repeated names are deleted in an mg-tree. At level t+1 or above, the correct vertex α has at least 2t+1 children, and out of which at least t+1 children are correct. The true values of these t+1 correct vertices are common, and the majority of the vertex value α is common. The correct vertex α is common in the mg-tree if the level of α is less then t+1. Consequently, all correct vertices of the mg-tree are common. ■

Lemma 8-2: The common frontier exists in the mg-tree.

Proof: By definition, an mg-tree is a tree of level t+1. There are t+1 vertices along each root-to-leaf path of an mg-tree. Since at most t processors can fail, there is at least one correct vertex along each root-to-leaf path of the mg-tree. Using Lemma 8-1, the correct vertex is common and the common frontier exists in each fault-free processor's mg-tree. ■

Lemma 8-3: Let α be a vertex, α is common if there is a common frontier in the sub-tree rooted at α.

Proof: If the height of α is 0 and the common frontier (α itself) exists, α is common. If the height of α is γ, the children of α are all in common under the induction hypothesis with the height of the children being γ-1. ■

Corollary 8-1: The value of root $\Re$ is common if the common frontier exists in the mg-tree.

Theorem 8-1: The value of root $\Re$ of a fault-free processor's mg-tree is common.

Proof: Using Lemmas 6-1, 6-2, 6-3 and Corollary 8-1, the theorem is proved. ■

Theorem 8-2: Protocol CP$_{p2p}$ solves the Consensus problem in a de-Bruijn overlay P2P file-sharing system.

Proof: To prove this theorem, CP$_{p2p}$ must meet the constraints (Consensus_Agreement) and (Consensus_Validity)

(Consensus_Agreement): Root value is common. By Theorem 8-1, (Consensus_Agreement) is satisfied

(Consensus_Validity): VOTE(α) = v for all fault-free processors, if the initial value of all processor is v_s say $v = v_s$.

Most processors are fault-free. The value of the correct vertices for all of the fault-free processors' mg-trees is v. Therefore, each correct vertex of the mg-tree is common (Lemma 8-1), and its true value is v. Using Theorem 8-1, this root is common. The computed value VOTE(α) = v is stored in the root for all the fault-free processors. Therefore, (Consensus_Validity) is satisfied. ■

8.5 Conclusion

The Consensus problem of file-sharing in P2P system becomes more complicated with the growing of P2P networks [2] [11] [17] [34] [20] [29] [35] [42] [49] [54] . The malicious processors could work in coordination with other malicious processors to modify files and spread inconsistent ones to other fault-free processors arbitrarily. If not properly controlled, fault-free processors may also spread inconsistent files to other processors and potentially paralyze the entire P2P network. So far, no previous has attempted to solve the Consensus problem of file-sharing with malicious processors in P2P networks.

In this chapter, we proposed a novel hybrid approach integrating the clustering algorithm and Consensus protocol to provide better QoS and solve the Consensus problem in the de-Bruijn overlay P2P file-sharing system. In the clustering algorithm, we adopt k-means algorithm to cluster similar processors into the same group to provide better QoS. Moreover, k-means algorithm is the simplest and most popular clustering algorithm [26] . Due to the fact that de-Bruijn graphs are nearly optimal [29] , we adopt de-Bruijn graphs as the overlay network in the P2P file-sharing system. We also proposed a Consensus protocol CP$_{p2p}$ get the consistent file information in the de-Bruijn overlay P2P file-sharing system.

Chapter 9
Conclusion and Future Work

9.1 Conclusion

In recent year, combined wired/wireless network have become more popular, the reliability and fault tolerance of combined wired/wireless network has become an important topic. We know that pure wired networks and pure wireless networks are all special cases of the combined wired/wireless networks. Hence, we also discuss the agreement problems in pure wired network and pure wireless network.

For pure wireless network, we proposed a BA protocol, Mobile Ad-Hoc Agreement Protocol (MAHAP), to solve the BA problem in wireless network with malicious faulty processors. MAHAP is the first BA protocol to solve the BA problem in MANET. The feature of mobility is considered in MAHAP.

For combined wired/wireless network, we proposed another BA protocol, Server-initiated Byzantine Agreement Protocol (SBAP), to solve the BA problem in combined wired/wireless network in the presence of malicious faulty processors. To meet the characteristics of mobile environments, most of the communication and computation overhead must be fulfilled within in the agreement-servers in SBAP. Furthermore, SBAP uses a hierarchical architecture to reduce the communication overhead. For combined wired/wireless network, we also proposed a Consensus protocol, Client-initiated Consensus Protocol (CCP), to solve the Consensus problem in combined wired/wireless network. Moreover, malicious fault assumption with processors grows into the dual failure mode (both dormant fault and malicious fault) on both processors and communication links.

In order to provide a highly reliable computing environment for combined wired/wireless network, we proposed a FDA protocol, Adaptive Fault Diagnosis Agreement Protocol (AFDA), to solve the FDA problem in combined wired/wireless network. AFDA is an adaptive FDA protocol. AFDA not only can solve the FDA problem in combined wired/wireless network, but also AFDA can solve the FDA problem in wireless network.

In the usage, the file-sharing application has been the most popular application in P2P systems. Many P2P networks are overlay networks because they run on top of the combined wired/wireless network. However, malicious attackers may modify files arbitrarily and spread inconsistent files to other processors in the P2P network. So far, no previous study has attempted to solve the Consensus problem of file-sharing with malicious processors in P2P networks. Hence, we give an application of Consensus protocol to ensure the file consistency of file-sharing in P2P networks.

9.2 Future Work

In this dissertation, the proposed BA and Consensus protocols required each fault-free processor to reach a common agreement at the same round of message-exchange. That is, even when the system has a smaller number of faulty processors, or there is no faulty processor at all in the system, the system still needs the same rounds of message exchange to reach a common agreement. The above problem is also called the Immediate Agreement Problem (IAP) [19] . As a result, the IAP has a consistent round complexity. However, the message-exchanging phase is a time-consuming phase, and the IAP does not seem efficient enough when the number of faulty processors is smaller than the tolerable number of faulty processors in a network. To improve the efficiency, another related problem called early stopping agreement problem (also called Eventual Agreement Problem, EAP) [15] [27] [58] can be visited. An early stopping agreement protocol is able to stop as early as possible when a processor receives enough information from other processors. Therefore, our future work will be focused on solving the early stopping problem in combined wired/wireless network.

Bibliography

[1] J. C. Adams and K.V.S. Ramarao, “Distributed diagnosis of Byzantine processors and links,” *Proceeding of the Symposium on Distributed Computing Systems*, pp.562-569, 1989.

[2] J. Aspnes, Z. Diamadi, and G. Shah, “Fault-Tolerant Routing in Peer-to-Peer Systems,” *in Proceeding of the 23rd Annual Symposium on Principles of Distributed Computing,* pp.223-232 ,2002.

[3] O. Babaoglu, and R. Drummond, “Streets of Byzantium: Network Architectures for Fast Reliable Broadcasts,” *IEEE Transactions on Software Engineering,* Vol. 11, no. 6, 1985, pp. 546-554.

[4] A. Bar-Noy, D. Dolev, C. Dwork and H. R. Strong, “Shifting Gears: Changing Algorithms on the Fly to Expedite Byzantine Agreement,” *Information and Computation*, vol.97, no.2, pp.205-233, 1992.

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

[6] B. Bellur and R. G. Ogier, A Reliable, “Efficient Topology Broadcast Protocol for Dynamic Networks,” *Proceeding of the 18th IEEE INFOCOM*, pp.178-186, 1999.

[7] W. G. Bridges and S. Toueg, “On the Impossibility of Directed Moore Graphs,” *Journal of Combinatorial Theory,* no. 3, 1980.

[8] D. R. Broug, Logic Programming. *New Frontiers*, Kluwer Academic, 1992.

[9] T. Camp, J. Boleng, and V. Davies, “A Survey of Mobility models for Ad Hoc Network Research,” *Wireless Communications & Mobile Computing*, vol. 2, no. 5, 2002.

[10] T. Chandra and S. Toueg, “Unreliable Failure Detectors for Reliable Distributed Systems,” *Journal of the ACM*, Vol. 43, No. 4, 1996, pp. 225-267.

[11] B. Cohen, “Incentives Build Robustness in BitTorrent,” *Proceeding of the Workshop on Economics of Peer-to-Peer Systems*, pp.1-5, 2003.

[12] G. Coulouris , J. D. Dollimore, T. Kindberg, “*Distributed Systems-Concepts and Design*”,3rd Edition, Addison-Wesley 2001.

[13] G. D. Crescenzo, R. Ge and G. R. Arce “Securing Reliable Server Pooling in MANET Against Byzantine Adversaries,” *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 2, 2006, pp. 357-369.

[14] J. Daemen, V. Rijmen, “*The Rijndael Block Cipher*,” AES Document Version2.

[15] D. Dolev, R. Reischuk, and A.R. Strong, “Early Stopping in Byzantine Agreement,”

ACM for Computing Machinery, vol. 37, no. 4, pp.720-741, 1990.

[16] W. Diffie and M. E. Hellman, “New Directions in Cryptography,” *IEEE Transactions on Information Theory*, vol.22, pp. 644-654, 1976.

[17] M. Feldman, K. Lai, I. Stoica, and J. Chuang, “Robust Incentive Techniques for Peer-to-Peer Networks,” *Proceeding of the 5th ACM Conference on Electronic Commerce,* pp.102-111, 2004.

[18] FIPS 180-1. *Secure Hash Standard.* U.S. Department of Commerce/NIST, National Tehnical Information Service, Springfield, VA, Apr. 1995.

[19] M. Fisher, and N. Lynch, “A Lower Bound for the Assure Interactive Consistency,” *Information Processing Letters,* vol.14, no.3, pp.183-186, 1982.

[20] Gnutella, http://www.gnutella.com

[21] R. Guerraoui and A. Schiper, “The Generic Consensus Service,” *IEEE Transactions on Software Engineering*, vol. 27, no. 1, 2001, pp. 29-41.

[22] X. Hong, K. Xu, and M. Gerla, “Scalable routing Protocols for Mobile Ad Hoc Networks,” *IEEE Network*, vol. 16, no.4, pp.11-21, 2002.

[23] H.S. Hsiao, Y.H. Chin, W.P. Yang, “Reaching Fault Diagnosis Agreement under a Hybrid Fault Model,” *IEEE Transactions on Computers*, vol. 49, no. 9, pp.980-986, 2000.

[24] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum and L. Viennot, “Optimized Link State Routing Protocol for Ad Hoc Networks,” *Proceedings of IEEE International Technology for the 21st Century*, 2000.

[25] D. B. Johnson and D. A. Maltz, *Dynamic Source Routing in Ad Hoc Wireless Networks, Mobile Computing*, Kluwer, 1996.

[26] L. Kaufman and P. J. Rousseeuw, finding Groups in Data: An Introduction to Cluster Analysis, *John Wiley & Sons, Inc.,* New York, 1990.

[27] A. W. Krings and T. Feyer, “The Byzantine Agreement Problem: Optimal Early Stopping,” *Proceedings of 32nd Hawaii International Conference on System Sciences*, LNCS 520, 1999.

[28] L. Lamport, R. Shostak, and M. Pease, “The Byzantine Generals Problem,” *ACM Transactions on Programming Languages and Systems*, vol.4, no.3, pp.382-401, 1982.

[29] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh, “Graph-Theoretic Analysis of Structured Peer-to-Peer Systems: Routing Distances and Fault Resilience,” *Proceeding of Applications, technologies, architectures, and protocols for computer communications,* pp.395-406, 2003.

[30] S. Mallela and G. M. Masson, “Diagnosable systems for intermittent faults,” *IEEE Transaction on Computers*, vol. 27, no. 6, pp. 560-566, 1978.

[31] N. Malpani, J. L. Welch and N. Vaidya, “Leader election algorithms for mobile ad hoc

networks," *Proceedings of the 4th International workshop on Discrete algorithms and methods for mobile computing and communications*, pp.96-103, 2000.

[32] S. Marano, V. Matta and L. Tong, "Distributed inference in the presence of Byzantine sensors," *Proceedings of IEEE Asilomar Conference on Signals, Systems, and Computers*, 2006, pp. 281-284.

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

[34] Napster, http://www.napster.com

[35] M. Naor and U. Wieder, "A Simple Fault Tolerant Distributed Hash Tables," *Lecture Notes in Computer Science,* vol. 2735, pp. 88-97, 2003.

[36] M.R. Pearlman and Z. J. Haas, "Determining the optimal configurations for the zone routing protocol," *IEEE Journal on Selected Areas in Communications*, vol.17, no. 8, pp.1395–1414, 1999.

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

[38] G. Pei, M. Gerla, and T.W. Chen, "Fisheye State Routing: A Routing Scheme for Ad Hoc Wireless Network," *Proceedings of the IEEE International Conference on Communications*, pp.70-74, 2000.

[39] C.E. Perkins, Ad Hoc Networking, Addison-Wesley, 2001.

[40] C.E. Perkins and E. M. Royer, "Ad-Hoc On-Demand Distance Vector Routing," *Proceedings of the 1st IEEE Workshop on Mobile Computing Systems & Applications*, pp.90–100, 1999.

[41] G. Rabbat, D. Nowak and A. Bucklew, "Generalized Consensus Computation in Networked Systems with Erasure Links," *Proceedings of IEEE 6th Workshop on Signal Processing Advances in Wireless Communications*, 2005, pp. 1088-1092.

[42] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Schenker, "A scalable content-addressable network", *in Proceeding of ACM SIGCOMM*, pp. 161-172, Aug. 2001.

[43] N. Roy, S. K. Das, K. Basu and M. Kumar, "Enhancing Availability of Grid Computational Services to Ubiquitous Computing Applications," *in Proceeding of 19th IEEE International Parallel and Distributed Processing Symposium*, pp.92a-92a, 2005.

[44] C. Santivanez, R. Ramanathan, and I. Stavrakakis, "Making Link-State Routing Scale for Ad Hoc Networks," *Proceedings of the 2nd ACM International. Symposium on Mobile Ad Hoc Net. & Compputing*, pp.22-32, 2001.

[45] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, John Whily & Sons, Inc. 1994.

[46] K. Shin and P. Ramanathan, “Diagnosis of Processors with Byzantine Faults in a Distributed Computing Systems,” *Proceedings of International Conference on Fault-Tolerant Computing,* pp.55-60, 1987.

[47] A. Silberschatz, P.B. Galvin, G. Gagne, *Operating System Concepts 6th Ed.,* John Wiley & Sons, Inc, 2002.

[48] T. Simunic “Power Saving Techniques for Wireless LANs,” *Proceedings of Design, Automation and Test in Europe*, vol. 3, pp. 96-97 2005.

[49] E. Sit, R. Morris, “Security Considerations for Peer-to-Peer Distributed Hash Tables,” *Proceedings for the 1st International Workshop on Peer-to-Peer Systems*, pp. 261-269, 2002.

[50] H. S. Siu, Y.H. Chin, W.P. Yang, “A Note on Consensus on Dual Failure Modes, *IEEE Transactions on Parallel and Distributed Systems*,” vol.7, no.3, pp.225-229, 1996.

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

[52] H.S. Siu, Y.H. Chin, and W.P. Yang, “Reaching Strong Consensus in the Presence of Mixed Failure Types,” *Information Sciences: An International Journal,* vol.108, nol.1-4, pp.157-180, 1998.

[53] K.N. Sivarajan and R. Ramaswami, “Lightwave Networks Based on de Bruijn Graphs,” *IEEE/ACM Trans. On Networking*, vol. 2, no. 1994.

[54] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," *in Proceeding of ACM SIGCOMM*, pp. 149-160, 2001.

[55] Y.F. Tsou, “ A particular Solution for Agreement Problem under Cluster-oriented MANET,” Master Thesis, Department of Information Management, Chaoyang Technology of University, Taiwan, 2007.

[56] J. Turek and D. Shasha, “The Many Faces of Consensus in Distributed Systems,” *IEEE Computer,* Vol. 25, No. 6, 1992, pp. 8-17.

[57] X. Wang and J. Cao, “An Optimal Early Stopping Uniform Consensus Protocol in Synchronous Distributed Systems with Orderly Crash Failure,” *Proceeding of the 23th International Conference on Distributed Computing Systems*, pp.76-81, 2003.

[58] S.C. Wang and C.F. Cheng, “Eventually Dual Failure Agreement,” in *Fundamenta Informaticae*, vol. 57, no. 1, pp.79-99, 2003

[59] S.C. Wang, Y.H. Chin, and K.Q. Yan, “Reaching a Fault Detection Agreement,” *Proceedings International Conference on Parallel Processing,* pp.251-258, 1990.

[60] S.C. Wang, Y.H. Chin, and K.Q. Yan, “Byzantine Agreement in a Generalized Connected

Network," *IEEE Transactions on Parallel and Distributed System,* vol.6, no.4, pp.420-427, 1995.

[61] S.C. Wang、K.Q. Yan and C.F. Cheng, "Achieving High Efficient Byzantine Agreement with Dual Components Failure Mode on a Multicasting Network," *Proceedings of the 9th IEEE International Conference on Parallel and Distributed Systems*, pp. 577-582, 2002.

[62] S.C. Wang、K.Q. Yan and C.F. Cheng, "Evidence-based MultiCasting Fault Diagnosis Agreement with Fallible Processors", *Proceedings of the 32nd International Conference on Parallel Processing*, pp.69-74, 2003.

[63] S.C. Wang, K.Q. Yan and H.C. Hsieh, "The New Territory of Mobile Agreement," *Computer Standards & Interfaces*, vol. 26, pp. 435-447, 2004.

[64] S.C. Wang, K.Q. Yan, H.C. Hsieh, "Reaching Consensus Underlying a Mobile Environment," *Proceeding of the 2003 Digital Life and Internet Technology Symposium*, 2003.

[65] S.C. Wang, J.E. Yang, K.Q. Yan, and C.F. Cheng, "Achieving High Efficient Consensus in a Hybrid Fallible Multicasting Network," *Proceedings of International Conference on Systems Engineering*, 2002.

[66] S.C. Wang, K.Q. Yan and G.Y. Zheng, "Dual Agreement Virtual Subnet Protocol for Mobile Ad-Hoc Networks," *Proceeding of the 22nd Annual ACM Symposium on Applied Computing*, pp. 11-15, 2007.

[67] S.C. Wang、W.P. Yang and C.F. Cheng, "Byzantine Agreement on Mobile Ad-Hoc Network," *Proceeding of the IEEE International Conference on Networking, Sensing and Control*, pp. 52-57, 2004.

[68] D. B. West, *Introduction to Graph Theory*, 2nd. Ed., Prentice Hall 2001.

[69] R. Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645-678, 2005.

[70] K.Q. Yan, Y.H. Chin and S.C. Wang, "Optimal Agreement Protocol in Malicious Faulty Processors and Faulty Links," *IEEE Transactions on Knowledge and Data Engineering*, vol.4, no. 3, pp.266-280, 1992.

[71] K.Q. Yan, S.C. Wang, Y.F. Tsou, "Revisit Consensus in a Dual Fallible Clustered-MANET," *Proceeding of the 2006 Taiwan Academic Network Symposium*, 2006.

[72] K.Q. Yan, S.C. Wang G.Y. Zheng, "Reaching Dual Fallible Virtual Subnet Consensus," *Proceeding of the 3rd International Conference on Soft Computing and Intelligent Systems and 7th International Symposium on advanced Intelligent Systems*, pp.20-24, 2006.