DESIGN AND ANALYSIS OF

ACCOUNTABLE NETWORKED AND DISTRIBUTED SYSTEMS

by

ZHIFENG XIAO

YANG XIAO, COMMITTEE CHAIR

MARCUS BROWN
XIAOYAN HONG
JINGYUAN ZHANG
SHUHUI LI

A DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the Graduate School of
The University of Alabama

TUSCALOOSA, ALABAMA

2013

ABSTRACT

This dissertation focuses on the design and analysis of accountable computing for a wide range of networked systems with affordable expense. The central idea is to incorporate accountability, a long-neglected security objective, into the design and implementation of modern computing systems. Broadly speaking, accountability in the cyber-security domain means that every entity ought to be held responsible for its behavior, and that there always exists undeniable and verifiable evidence linking each event to the liable entities. This dissertation studies accountable computing in three different contexts, including traditional distributed systems, cloud computing, and the Smart Grid.

We first propose a quantitative model called P-Accountability to assess the degree of system accountability. P-Accountability consists of a flat model and a hierarchical model. Our results show that P-Accountability is an effective metric to evaluate general distributed systems such as PeerReview [1] in terms of accountability. Next, we develop Accountable MapReduce for cloud computing to prevent malicious working machines from manipulating the processing results. To achieve this goal, we set up a group of auditors to perform an Accountability-Test (A-test) that checks all working machines and detects malicious nodes in real time. Finally, we investigate the accountability issues in the neighborhood area smart grid. A mutual inspection scheme is presented to enable non-repudiation for metering. In addition, we propose and analyze a suite of algorithms to identify malicious meters for the detection of energy theft.

DEDICATION

This dissertation is dedicated to everyone who helped me, guided me, and inspired me.

LIST OF ABBREVIATIONS

ACK	Acknowledgement
AD	Administrative Domain
AG	Auditor Group
AIP	Accountable Internet Protocol
AMI	Advanced Metering Infrastructure
A-test	Accountability test
ATI	Adaptive Tree Inspection
BI	Binary Inspection
CA	Certificate Authority
CATS	Certified Accountable Tamper-evident Storage
CSAR	Cryptographically Strong, Accountable Randomness
DBI	Dynamic Binary Inspection
DFS	Distributed File System
DoS	Denial of Service
E2E	End-to-End
GH	Group Head
GM	Group Member
HAN	Home Area Network
ISP	Internet Service Provider

LAN	Local Area Network
MAC	Message Authentication Code
MITM	Man-In-The-Middle
MLP	Message Loss Probability
MMI	Malicious Meter Inspection
NAN	Neighborhood Area Network
NIC	Network Interface Card
OWAA	Optimal Worker and Auditor Assignment
P2P	Peer-to-Peer
PAN	Personal Area Network
PKI	Public Key Infrastructure
PM	Perfect Mapping
Re-ECN	Re-Explicit-Congestion-Notification
RPC	Remote Procedure Call
SLA	Service Level Agreement
SSL	Security Socket Layer
SVM	Support Vector Machine
TPR	Traceable PeerReview
TWM	Time Window Maximization
VLAN	Virtual Local Area Network
WAN	Wide Area Network
WSN	Wireless Sensor Network

ACKNOWLEDGMENTS

I am pleased to have this opportunity to express my appreciation to the faculty members, family, colleagues, and friends who have been offering their support and help throughout the 5-year course of my Ph.D. study.

First of all, I would like to thank my advisor, Dr. Yang Xiao, for his professional training and continuous encouragement over the past five years. There is no way I can sharpen my research skills and widen my scientific vision without the hundreds of research meetings and technical discussions with Dr. Xiao. Apart from that, Dr. Xiao has been influencing me with his enthusiasm on research and teaching, his rigorous attitude toward details, and his persistence and diligence on work.

Besides my advisor, I would also like to thank the rest of my committee members, Dr. Marcus Brown, Dr. Xiaoyan Hong, Dr. Shuhui Li, and Dr. Jingyuan Zhang, for their invaluable comments, inspiring questions, and constructive suggestion of both the dissertation and my academic progress.

This research would not have been possible without the support and encouragement of my family. My deepest appreciation goes to my parents for giving birth to me and supporting me spiritually throughout my life. Last but not the least, I am particularly grateful to my wife for her constant love and company.

This work is supported in part by the US National Science Foundation (NSF) under the grant numbers CNS-0737325, CNS-0716211, CCF-0829827, and CNS-1059265.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Undoubtedly, cyber-security is one of the top priorities for modern information systems. It is not rare that a small design flaw can lead to severe system vulnerability, which may be further exploited by attackers or adversaries. Recent data breach incidents have again proven the importance of security. With the development of new computing paradigms, new security solutions are highly required to deal with the state-of-the-art security demands and challenges.

This dissertation focuses on providing an accountable environment for a wide range of computer or networked systems with affordable expense. The central idea is to incorporate accountability, a long-neglected security property, into the design and implementation of current computing systems. In the security domain, broadly speaking, accountability means that every entity ought to be responsible for its behavior, and that there always exists undeniable and verifiable evidence to link each event with the liable entities. Numerous security issues attribute to the lack of accountability. For instance, IP forgery enables a malicious device to send packets with another identity without being detected. In this case, an accountable address is needed to ensure every device is held responsible for its IP address.

However, applying accountability to practical problems is not a simple task. First, accountability has different meanings in different environments, because an entity could be a lot of things, and its behavior varies accordingly. To this end, in a specific context, one should identify the actual definition of accountability by clarifying three terms: Entity, Event, and Evidence, namely, “3E” in short. Another challenge is how to generate evidence that points to

the liable entity. The third consideration is performance. A practical system does not allow serious performance degradation when additional security services are enabled. Accountability schemes (e.g., auditing, secure logging, cryptographic protocol) usually incur significant computation, storage, or communication overhead, which will slow down the host system. Therefore, to reach an adequate effect, a trade-off issue needs to be studied.

In this dissertation we will study accountability in three contexts, including traditional distributed systems, cloud systems, and the Smart Grid. Each environment poses particular research issues and challenges. Although it is hard to design a universal accountability scheme that applies to all systems, it is viable to develop a systematic method to evaluate the degree of accountability under different computing environments. Our first work presents a quantitative model named P-Accountability that provides both empirical and analytical metrics for accountable systems. We then propose certain techniques to implement accountable computing for the MapReduce system in the cloud setting, and for the Smart Grid in the Neighborhood Area Network (NAN). Our experiment results have shown the effectiveness of the proposed schemes in terms of the trade-off between accountability and performance.

1.1 Research Issues

1.1.1 A Performance Metric for Accountability

Accountability has been a longstanding concern of trustworthy computer systems [2], and it has recently been elevated to a first class design principle for dependable networked systems [2, 15]. Some studies [1, 3-5, 10-14] focus on the design of accountable systems for different applications. The notion of accountability in each of them varies. In a complex networked environment, the responsible party could be any entity (e.g., a web/database server, a mobile

device, a router/switch, a program running on a computer, etc). However, no matter what the specific definition is, the core ideas of accountability are 1) to assign responsibility with irrefutable evidence, and 2) to punish the responsible entities for their misbehavior, if necessary. Based on the literatures we have investigated, how to assess the degree of accountability remains to be an open problem. In this research, we attempt to answer a few questions:

- How accountable can a system can?
- What factors are there to affect the degree of accountability?
- What would it take to enhance system accountability?
- How does accountability affect system performance?

Motivated by these questions, we propose a quantitative model called P-Accountability, which provides both empirical and analytical metrics to assess accountability.

1.1.2 Issues in Cloud MapReduce

MapReduce [32] has been widely used as a powerful model for data processing. It has efficiently solved a wide range of large-scale computing problems, including distributed grep, distributed sort, web-link graph reversal, web-access log statistics, document clustering, etc. Recent study shows that a joint effort of MapReduce and Cloud Computing presents a promising technique for big data processing [44]. For example [15], the New York Times rented 100 Amazon EC2 instances and a Hadoop [33] application to convert 4TB of raw image TIFF data (stored in Amazon S3) into 11 million PDFs in 24 hours at a computation cost of $240 (not including bandwidth).

In such a situation, cloud customers outsource their data or computational tasks to the cloud. Therefore, trust becomes an issue. For cloud vendors, it is challenging to convince customers that their services are trustworthy. Some cloud machines may become malicious due

to an attack or infection. These compromised machines may tamper with the processing results, making MapReduce untrustworthy. In the New York Times example, if some malicious nodes tamper with the image conversion process, the resulting PDFs do not match the original TIFF images; and even worse, it is very difficult for the New York Times to check the correctness. To this end, we propose Accountable MapReduce which is capable of detecting dishonest cloud servers.

1.1.3 Issues in The Smart Grid

The Smart Grid has received significant attention in recent advances [48 - 50, 56, 60, 61]. A smart grid enables two-way flows of electricity and information to create an automated and distributed energy delivery system. Meanwhile, the Smart Grid security is becoming a significant concern. New infrastructure (e.g., advanced metering system and digital networks) requires new hardware and software, which introduce vulnerabilities. One of the serious issues is energy theft, which has been a chronic problem and still exists in the Smart Grid. Although some attacking methods in traditional power grids are not applicable to the Smart Grid, new methods have been developed to compromise its subsystems. For instance, researchers have successfully hacked into a smart meter with a recovered backdoor account [30], and then tampered with the meter reading. Therefore, it is imperative to develop countermeasures to combat energy theft in the Smart Grid.

1.3 Organization

The rest of the dissertation is organized as follows. Chapter 2 presents P-Accountability, which defines a quantitative model to assess the degree of accountability for distributed systems. Chapter 3 gives the design and analysis of an accountable cloud MapReduce system. Chapter 4

discusses the issue of non-repudiation on meter reading between customer and utility provider in the neighborhood area Smart Grid. Chapter 5 further studies the problem of energy theft through a suite of inspection algorithms. Chapter 6 concludes the dissertation.

1.4 Publication

This dissertation has led to publications, which are listed in Table 1.1.

Table 1.1 Publication Records

Chapter 2	Conference	• Zhifeng Xiao and Yang Xiao, "P-Accountable Networked Systems," IEEE INFOCOM 2010 Work In Progress Track, San Diego, CA, Mar. 2010. • Zhifeng Xiao, Yang Xiao, and Jie Wu, "A Quantitative Study of Accountability in Wireless Multi-hop Networks," 39th International Conference on Parallel Processing (ICPP), San Diego, CA, pp. 198 - 207, Sep 2010.
	Journal	• Zhifeng Xiao and Yang Xiao, "PeerReview Re-evaluation for Accountability in Distributed Systems or Networks," International Journal of Security and Networks, Vol. 7, No. 1, pp. 40-58, 2012. • Zhifeng Xiao, Yang Xiao, and Jie Wu, "P-Accountability: A Quantitative Study of Accountability in Networked Systems," Submitted to IEEE Transactions on Parallel and Distributed Systems.
Chapter 3	Conference	• Zhifeng Xiao and Yang Xiao, "Accountable MapReduce in Cloud Computing," 2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 1082 -1087, 2011.
	Journal	• Zhifeng Xiao and Yang Xiao, "Security and Privacy in Cloud Computing," IEEE Communications Surveys and Tutorials, Vol. 15, Issue 2, pp. 843 - 859, 2012. • Zhifeng Xiao and Yang Xiao, "Achieving Accountable MapReduce in Cloud Computing," Submitted to the journal of (Elsevier) Future Generation Computer Systems.
Chapter 4	Conference	• Zhifeng Xiao, Yang Xiao, and David Hung-Chang Du, "Building Accountable Smart Grids in Neighborhood Area Networks," 2011 IEEE Global Telecommunications Conference, 2011, pp. 1-5.
	Journal	• Zhifeng Xiao, Yang Xiao, and David Hung-Chang Du, "Non-repudiation in Neighborhood Area Networks for Smart Grid," IEEE Communications Magazine, Vol. 51, Issue 1, 2013, pp. 18 - 26.
Chapter 5	Journal	• Zhifeng Xiao, Yang Xiao, and David Hung-Chang Du, "Exploring Malicious Meter Inspection in Neighborhood Area Smart Grids," IEEE Transactions on Smart Grid, Vol. 4, Issue 1, 2013, pp. 214 - 226.

CHAPTER 2

P-ACCOUNTABILITY: A QUANTITATIVE STUDY OF ACCOUNTABILITY IN NETWORKED SYSTEMS

2.1 Introduction

Accountability has been a longstanding concern of trustworthy computer systems [2], and it has been recently elevated to a first class design principle for dependable networked systems [3-4]. Accountability implies that an entity should be held responsible for its own specific actions or behaviors so that it can be part of a larger chain of accountability [5]. Previous studies [1, 6-15] have discussed the design of accountable systems in different contexts. The notion of accountability in each of them varies. Through the literatures we have investigated, we observe that no matter what the specific definition is, the core ideas of accountability are 1) to assign responsibility with irrefutable evidence, and 2) to reward or punish the responsible party, if necessary. In a complex network environment, however, the responsible party could be any entity (e.g., a web/database server, a Personal Computer (PC), a mobile device, a router/switch, a program running on a computer, etc).

This work is motivated by two factors. First, if accountability is regarded as a system service and must meet a certain design requirement, it is imperative to develop a general metric to evaluate the degree of accountability. As feedback, the metric should be able to provide valuable guidance to improve the system design. Second, it is usually expensive to design a system with perfect accountability because tough conditions that a realistic system is unable to

afford may apply. For example, these conditions may come from accountability schemes, including strong identification for every host [1, 7, 8], a per-message/hop verification scheme [7], and a powerful logging/repository system [5, 13]. Applying the schemes in practice incur computation, storage, and communication overhead that affect system performance. In addition, it is difficult to achieve perfect accountability due to network dynamics such as packet loss, delay, and node failure. To this end, instead of pursuing perfect accountability, system designers only need to provide satisfactory accountability that will not over-consume system resources. Therefore, it is essential to study the balance of accountability and system overhead. To achieve this goal, we need to quantify accountability.

Our needs come down to one question: how accountable can a system be? In this chapter, we develop P-Accountability, a generic model to assess accountability for networked systems. P-Accountability models an accountable networked system by abstracting the notions of entities, events, and the mapping relation between them. P-Accountability should be customized to suit the needs for a practical system. To do that, we first need to identify the event space and entity space for a system, and then find out the factors that may affect the degree of accountability in order to give formal analysis and empirical study.

The contributions of this chapter include:

1. We propose P-Accountability, which is a quantitative model for accountability assessment. The model includes two parts: a flat model and a hierarchical model. The flat model is applicable to accountable systems with flat structures, and this means the entities are on the same logic level. We then extend the flat model into a hierarchical model, which considers a multi-level environment where each entity in a certain level may be further composed of fine-grained entities in a

lower level. This indicates that blame may be assigned to a more concrete entity in some circumstances.

2. We provide a complete case study to apply P-Accountability to PeerReview, which studies the Byzantine fault detection problem for distributed systems. We discover that message loss is a key factor affecting PeerReview's accountability. We demonstrate that P-Accountability can be successfully adopted to assess PeerReview-enabled systems.
3. With the feedback we obtain from the case study, we propose and analyze Traceable PeerReview (TPR) which extends PeerReview to a wireless multi-hop network environment.
4. Our evaluation results show that P-Accountability is effective in the assessment of accountability.

2.2 Related Work

2.2.1 Accountable Systems

Recent advances have sought to apply accountability to various network contexts. Yumerefendi *et al*. present a Certified Accountable Tamper-evident Storage service (CATS) [13], which is an application level storage service providing verifiable misbehavior detection. AudIt [12] is described as an accountability interface that facilitates Internet Service Providers (ISPs) to determine the Administrative Domains (ADs) that are responsible for dropping or delaying the traffic; AudIt can be implemented with a NetFlow modification and deployed in a border router to monitor inter-AD links. Andersen *et al*. present Accountable Internet Protocol (AIP) [6, 7], which employs a hierarchy of self-certifying addresses to enable network-layer accountability.

AIP requires the modification of the current IP protocol to implement a self-certifying addressing scheme. The authors also justify the feasibility of AIP in terms of scalability and performance. As an alternative to the operational accountability studied in AIP, Mirkovic *et al.* propose a perfect accountability scheme [8] by binding an unforgeable signature to each packet and having the closest router verify it. PeerReview [1] presents an accountable distributed system in a deterministic environment for Byzantine fault detection. Cryptographically Strong, Accountable Randomness (CSAR) [9] extends PeerReview's work to make randomized systems accountable. Keller *et al.* [10] have discussed the issue of accountability in hosted virtual networks and have examined two possible approaches. The first approach leverages network measurement techniques to detect violations of Service Level Agreements (SLAs), and the second approach re-architects a router to prevent SLA violation beforehand. Liu *et al.* [11] have proposed using a layered trust management framework to help email receivers eliminate their unwitting trust and provide them with accountability support. Re-Explicit-Congestion- Notification (Re-ECN) [14, 15] allows accurate congestion monitoring throughout the network, thus enabling the upstream party at any trust boundary in the Internet to be held responsible for the congestion that it causes or allows to be caused; Re-ECN requires to change the current TCP. Zeng *et al.* [16] has proposed accountable administration for operating systems, and developed a prototype in the Linux platform. Xiao *et al.* has proposed Flow_Net [5, 31] which is a novel logging mechanism that captures events and the relations among them for system accountability. Flow_Net can be implemented as a OS kernel service to capture, log, and audit events with multiple granularities.

Realistic deployment and implementation of accountability in the real world has not been widely conducted yet. Some systems [1, 9, 13, 16] have already been experimented with; nevertheless, regarding performance evaluation, they are more concerned with general system

performance metrics, such as message overhead, delay, and throughput, etc. The degree of accountability has not been well evaluated as a metric.

2.2.2 Theoretical Definitions of Accountability

Security quantification [17 - 19] has been studied in recent years. Formal definitions of accountability have existed in previous research [20, 22 - 24]. However, most prior studies attribute the degradation of accountability to internal problems such as cryptographic design flaws. Our model focuses on external and practical factors (e.g., network dynamics). These factors may not be considered when the protocol is designed in the first place, but they may become significant when the system is tested in the real world.

Bella and Paulson [20] develop a model to verify accountability protocols by employing an inductive method [21]. Two factors, validity of evidence and fairness, are involved to verify the accuracy of accountability protocols. The model is applied to the analysis of a certified email protocol and a non-repudiation protocol.

Jagadessan *et al*. [22] present an operational model of accountability based upon Communicating Sequential Processes, I/O automata, and discrete timed process algebra. With the model, auditors are able to blame message senders whose behaviour deviates from the pre-defined protocol specification. The problem of this framework is that it does not provide a mechanism to blame dishonest parties. In addition, it is unable to deal with cryptography.

Küsters *et al*. [23] discuss a general definition of accountability, which is built upon π-calculus and I/O automata with interpretations in both symbolic and computational models. Informally, the definition of accountability implies two properties: 1) fairness – no honest participants will be blamed; 2) completeness – the participants who misbehaved, or at least some

of them, will be blamed. The framework is applied to three protocols: contract-signing, voting, and auction.

Feigenbaum *et al*. [24] propose a formal accountability model based upon event traces and utility functions. This model takes anonymity into account for the first time. It argues that in some situations, an honest user should be able to remain anonymous, while a dishonest one who violates the security policy will be exposed.

2.3 A Flat Model for P-Accountability

We use the following notation style in this chapter: an uppercase letter, such as *V* and *E*, stands for a set, and a lowercase letter, such as *v* and *e*, stands for an element of a set.

2.3.1 A Flat Model

Based on the previous accountable systems that we investigated, we present a flat model for accountability. In most prior studies in [1, 6, 7, 9, 11, 12], the major function of accountability is blame assignment. However, the entities and events in different contexts differ. In general, we can model a networked system as $Q=(V,E)$, where V = {v | v is an entity in the system}, and E = {e | e is an event caused by entities in V}. Essentially, an accountable system will link an event to the responsible entity (or entities), and the process can be modeled as a mapping operation $\alpha: E \rightarrow \{V_e \mid V_e \subseteq V, e \in E\}$. Given an event e, $\alpha(e)$ will return the entity (or entities, denoted by V_e) generating (or causing) event e. However, in a practical system, the mapping function $\alpha(e)$ may not always be able to return a complete set of liable entities due to external factors. We define $\alpha(e)$ as a *perfect mapping (PM)* if and only if "$\alpha(e)=V_{e|PM} \subseteq V$ and $V_{e|PM}$ is the complete set of correct results". Note that if $\alpha(e)$ is not a PM, then $\alpha(e)=V_e \subset V_{e|PM} \subseteq V$. For example, a traffic flow is delayed or lost by multiple administrative

domains (ADs) including AD1, AD2, and AD3; but the accountable system is only able to find AD1 and miss the other two. The mapping in this example is correct, but not perfect.

Definition 2.1: *Accountability (flat model)*: in a networked system $Q=(V,E)$, for $\forall e\in E$, if $\alpha(e)$ is a PM, then the system is accountable; otherwise the system is non-accountable. In other words, accountability $\Psi(Q)$ is defined as

$$\Psi(Q)=\prod_{e\in E} I(\alpha(e) \text{ is a PM}) \tag{2.1}$$

where *I(x)* is an indication function, which returns 1 if *x* is true and 0 otherwise.

Definition 2.1 makes accountability a binary value. Being 1 implies that a system is perfectly accountable, and being 0 means the system is not accountable at all. In reality, however, a system might not be able to achieve perfect accountability, meaning that the mapping can be correct, but not perfect. Therefore, we need a quantity to evaluate accountability. We propose P-Accountability for this purpose. "P" stands for *performance* and *probability*, because P-Accountability provides not only a performance metric for empirical study, but also a probabilistic approach for system analysis.

Definition 2.2: *P-Accountability (flat model)***:** in a networked system *Q = (V, E)*, P-Accountability $A_F(Q)$ is defined as follows:

$$A_F(Q)=\frac{1}{|E|}\cdot\sum_{e\in E}\left(\frac{|V_e|}{|V_{e|PM}|}\cdot I(\alpha(e) \text{ is correct})\right) \tag{2.2}$$

where operation $|set|$ computes the size of a set. We can easily show that the definitions 2.1 and 2.2 are consistent when the system is perfectly accountable, i.e., for $\forall e\in E$, $I(\alpha(e) \text{ is a PM})\equiv 1$. After we have the definition of P-accountability in (2.2), the definition of accountability in (2.1)

will not be used. Let $P(e)$ be the probability of $\alpha(e)$ being a PM. We can use a probabilistic approach to estimate the value of P-accountability as follows

$$A_F(Q) \approx \bar{A}(Q) = P(e)$$

2.3.2 Usage of the Flat Model

The flat model is applicable to an accountable system with flat structure, i.e., all entities are homogeneous. In a particular context, entities and events may vary. For example, AIP blames a host with a fake IP address, thus the basic entities are network devices with IP addresses, and an event could be "host A received a message from host B [192.168.47.128]"; AIP answers the question that "is IP [192.168.47.128] the real message source?" The flat model can also be used on AudIt. Traffic flowing through an AD can be relayed successfully, dropped, or delayed. Accountability in AudIt implies that each AD is responsible for the traffic that it has dealt with. Therefore, the basic entities are ADs, and the events are incidents of traffic loss/delay/drop. For any accountable system with flat structure, once the entity space and event space are determined, we can apply the flat model of P-Accountability to evaluate system accountability. We conduct a complete case study that applies P-Accountability to PeerReview in Section 2.5.

2.4 A Hierarchical Model for P-Accountability

The definition of P-Accountability in Section 2.3 is only applicable to flat network model. In this section, we consider a more complex network environment, which is comprised of multiple hierarchies. We then develop a hierarchical model for P-Accountability as an advanced metric to quantify accountability.

2.4.1 A Hierarchical Definition of Accountability

A networked system is made up of various physical devices and software elements. Logically, a network can be viewed hierarchically. Each hierarchy is comprised of one type of

entity. Table 2.1 shows an example with 5 hierarchies, in which the top hierarchy (i.e., H_1) is the whole network. The one next to it is H_2, which consists of sub-networks/domains. In this chapter, H_2 is loosely defined so that all sub-networks/domains are placed into H_2, regardless of the internal relationships among them. H_3 is made up of networking devices. H_4 contains diverse network applications, services, and programs that are running on the devices in H_3. Note that there might be multiple services running on one device. The bottom hierarchy, H_5, contains many kinds of conversational elements, including packets, traffic streams, messages, etc. These elements are generated by entities from the upper hierarchy; they carry specific information, and can be processed by the same kind of entities on the other end of the network. The hierarchies are not invariable and can be built selectively based on the system to be evaluated. For example, when we study a small scale network (e.g., Personal Area Network (PAN), Local Area Network (LAN)), H_2 can be removed. Also, for a simple embedded network system, we can ignore H_4 because each device (e.g., a temperature sensor node) in such a system may only run one specific application to fulfill its assigned tasks. With hierarchies, a more fine-grained model of accountability is given. For instance, to defend against a Denial of Service (DoS) attack, the network administrator attempts to identify the source of the malicious traffic, which might be generated by some program (H_4) on multiple computers (H_3) located within different domains (H_2). In reality, not every attack source can be determined. In addition, the accuracy of accountability differs. Intuitively, the deeper the attack source can be positioned in the hierarchy, the more accountable the system is. For example, an event could be "a malicious program *x* running on a computer *y* located in a university *z* launches a DoS traffic," meaning that *x* is the root cause entity responsible for this event. A system with weak accountability may only be able

to trace the event back to *z*, or a department of *z*, while a system with strong accountability is able to pinpoint the program *x* on *y*. Apparently, the latter system is more accountable.

Table 2.1 An example (5-level) of A hierarchical structure of network systems

Hierarchy	Description	Entities
H_1	entire network	Internet, LAN, WAN, PAN, WSNs, etc.
H_2	Sub-networks	Federal departments, enterprises, universities, other administrative domains, etc.
H_3	Devices	PC, Laptop, router, Server, PDA, etc.
H_4	Applications	Services, programs, protocols, etc.
H_5	Conversational elements	Messages, packets, traffic flows, etc.

In general, we assume a system has n hierarchies. Let H_i $(1 \le i \le n)$ denote a certain hierarchy. To be consistent, we still let V and E denote the entity set and event set, respectively. However, in this context, the cause of an event has granularities, involving multiple entities at different hierarchies. The definition of mapping function α does not change, but the result entity set V_e should locate at a certain hierarchy. In addition, the possible correct results of $\alpha(e)$ might locate in different hierarchies, if multiple granularities are considered. For example, if an event e is caused by v, which is a program (v locates at H_4) running on computer $v' \in H_3$, then both v and v' are correct results. Obviously, v has finer granularity than v'. A more accountable system has a higher chance to find a correct result with finer granularity. Therefore, the notion of perfect mapping is redefined as "for $e \in E$, the mapping result V_e is correct, complete, and deepest." A result is deepest means that the system is unable to find a correct result with finer granularity.

Definition 2.3: *P-Accountability (hierarchical model)*: for a networked system $Q = (V, E)$, P-Accountability $A_H(Q)$ is defined as follows:

$$A_H(Q) = \frac{1}{|E|} \cdot \sum_{e \in E} \left(\frac{d_e}{d_{e|PM}} \cdot \frac{|V_e|}{|V_{e|C}|} \cdot I(\alpha(e) \text{ is correct}) \right) \tag{2.3}$$

In the above definition, d_e and $d_{e|PM}$ denote the indices of the hierarchies where V_e and $V_{e|PM}$ locate at, respectively. $V_{e|C}$ denotes the complete and correct result set at hierarchy d_e. Obviously, if V_e lies in a higher hierarchy than $V_{e|PM}$, then event e is less accountable since existing evidence is unable to generate a PM that points to the root causal entities.

2.4.2 Example: P-Accountability with AudIt

AudIt enables ISPs to proactively supply feedback on traffic loss/delay events. We can apply the hierarchical model to AudIt as well. The ADs in AudIt belong to H_2. The traffic source could be an AD/ISP (H_2), a PC (H_3), or a specific service (H_4). The major event generated by ADs is traffic relay. According to the notion of hierarchical accountability, a more fine-grained responsibility can be assigned to a specific router in the AD or even to a particular network interface that is responsible for the event of packet dropping or delaying. In addition, $\bar{A}(Q)$ can be defined as the probability that a traffic drop/delay event can be correctly traced back to a responsible Network Interface Card (NIC).

2.5 Applying P-Accountability to PeerReview

2.5.1 PeerReview Overview

PeerReview [1] is a software library which ensures that 1) Byzantine faults whose effects are observed by a correct node are eventually detected and irrefutably linked to a faulty node, and that 2) a correct node can always defend itself against false accusations. Some critical assumptions in PeerReview are listed as follows:

- The application that each node runs is deterministic, meaning that a certain input will always return the same result regardless of the system state. To check the correctness of other nodes, a reference implementation is employed. For a specific

input, only the output produced by the reference implementation is correct, and any other different outputs are incorrect.

- A message sent from one correct node to another is eventually received, if retransmitted sufficiently often.
- Each node has a public/private key pair bound to a unique node identifier, and no identity is forgeable.
- Each node is indicated as either 'trusted', 'suspected', or 'exposed'.
- Each node *k* has a witness set $\omega(k)$ which will monitor *k*'s behavior and inform other nodes if *k* becomes abnormal.

PeerReview consists of six components: tamper-evident logs, commitment protocol, consistency protocol, audit protocol, challenge/response protocol, and evidence transfer protocol. Based on our studies [25, 28, 34], PeerReview is unable to achieve perfect accountability in a complex network environment (e.g., the Internet) due to end-to-end packet dynamics [27]. In addition, false accusations may happen because messages could be eventually lost.

2.5.2 Network Model

The flat model of accountability suits PeerReview very well because there is merely one hierarchy that contains all nodes (i.e., hosts) in a distributed system. Before further analysis is given, we introduce some notations:

Q(V, E) - A PeerReview-enabled system, in which *V* stands for the entity set (i.e., computers), and *E* stands for the event set, which will be defined later in this section.

V_H - The set of honest nodes. We have $V_H \subseteq V$.

φ - The fraction of faulty nodes. φ could be very small in the beginning, and increases during the running time because some nodes may get compromised.

w - The witness size. In this chapter, we assume that the witness size for all nodes is constant, i.e., $\forall v \in V$, $|\omega(v)| = w$.

2.5.3 P-Accountability on PeerReview

In order to evaluate the system accountability for PeerReview, we first examine the possibilities that a node judges another one, as well as the reason behind that. We define a status set O = {Correct (C), Faulty (F), and Ignorant (I)} for nodes in PeerReview. Let o_v represent the status of node *v*, such that $o_v \in O$. According to PeerReview [1], an *ignorant* node can be loosely defined as a node that ignores all incoming messages and does not respond to any node. A *faulty* node is a node that suffers from a general Byzantine fault, thus its behavior becomes arbitrary and unpredictable. Strictly speaking, an ignorant node is also a faulty node, but the system itself is unable to tell the difference between an ignorant node and a node suffering from severe message loss. Thus no evidence can be provided to expose an ignorant node. A correct node faithfully complies with the application protocol, meaning that it always responds as desired.

An indication of node *i* is how node *i* will be judged by other nodes. In the PeerReview context, the indication set is defined as U = {*T*, *X*, *S*}, where *T*, *X*, and *S* stand for 'trusted', 'exposed', and 'suspected', respectively. For $\forall v \in V$, an indication table $\Gamma_v = \{< i, u_{i,v} > | i \in V, u_{i,v} \in U\}$ is maintained in node *v*, where $u_{i,v}$ is an indication of node *i* generated by node *v*. If $u_{i,v}$ = *T*, node *v* considers node *i* as a trusted node and interacts with node *i* according to normal procedure. In particular, we have $u_{v,v} \equiv T$, i.e., node *v* always considers itself as a trusted node. If $u_{i,v}$ = *X*, node *v* considers node *i* as an exposed node and stops communicating with *i* so that *v* is protected from deception. If $u_{i,v}$ = *S*, node *v* considers node *i* as a suspected node. Initially, we

let $\Gamma_v = \{< i, u_{i,v} > | \forall i \in V, u_{i,v} = T\}$. In other words, node *v* will treat every other node as a trusted node in the beginning. When the system is running, Γ_v will be updated in real time.

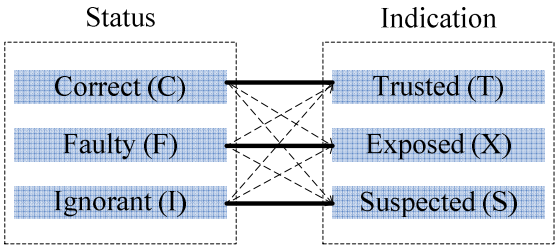


Fig. 2.1 Statuses and Indications

Fig. 2.1 shows the combination of statuses and indications. Let $u_{i,v} : o_i$ denote the indication of node *i* generated by node *v*, given that node *i* is in status o_i. For instance, $(u_{i,v} = T) : (o_i = C) = T : C$ means that node *i* is 'Trusted' by node *v* when node *i* is a correct node. For $\forall v, i \in V$, obviously, there are nine possible ways, shown in Table 2.2, for node *v* to indicate any possible status of node *i*.

Table 2.2 Nine indication possibilities of $u_{i,v} : o_i$

Accurate indication	Error
T:C, X:F, S:I	*X:C, S:C, T:F, S:F, T:I, X:I*

From Table 2.2, node *v* accurately indicates the status of node *i* in three out of nine cases, and the other six cases are errors (i.e., inaccurate indications). For PeerReview, message loss is considered to be the only cause of errors. If message loss never happens, the network is error free and completely accountable. PeerReview allows message loss, but it assumes that a temporarily lost message will eventually arrive at the destination, hence errors could happen in a short term but will ultimately be avoided. Therefore, PeerReview is able to achieve percent accountability eventually. According to the primitives of PeerReview, event *X:I* will not happen because a

correct node is unable to provide any evidence to expose an irresponsive node. Even under our assumption that messages may eventually be lost, the situation of *X:I* will also not happen.

Definition 2.4: *False positive* includes: 1) A correct node falsely 'trusts' a faulty node, i.e., *T:F*; 2) A correct node falsely 'trusts' an ignorant node, i.e., *T:I*.

Definition 2.5: *False negative* includes: 1) A correct node falsely 'exposes' a correct node, i.e., *X:C*; 2) A correct node falsely 'suspects' a correct node, i.e., *S:C*.

We define the event space in PeerReview as $E=\{(i,j) \mid \forall i \in V, \forall j \in V_H$, node *i* is indicated by node *j* $\}$. If node *i* is correctly indicated by node *j*, then we count this indication as a *PM*. The size of event set is thus $|E|=|V| \cdot |V_H|$, Based on definition 2.2, we obtain P-Accountability in PeerReview as follows:

$$A_F(Q) = \frac{\sum_{v \in V} \left(\text{Number of PMs at node } v \right)}{|V| \cdot |V_H|} \tag{2.4}$$

For $\forall v \in V_H$, let P_C denote the probability that node *v* correctly indicates the status of any other node in *V*. Let P_{FP} and P_{FN} denote the probabilities of false positive and false negative, respectively. We obtain the probabilistic model of P-Accountability $\bar{A}(Q)$ for PeerReview as follows.

$$\bar{A}(Q) = P_C = 1 - P_{FP} - P_{FN} \tag{2.5}$$

Equation (2.5) is consistent with equation (2.3). In order to calculate P_C we first calculate the End-to-end (E2E) Message Loss Probability (MLP) and the eventual MLP, and then obtain P_{FP} and P_{FN}.

1)E2E MLP and eventual MLP

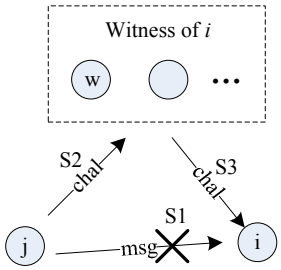


Fig. 2.2 Communication in PeerReview

In PeerReview, a temporary message loss event does not imply an eventual message loss because the challenge-response protocol will help the destination node receive the lost message at a later moment through a challenge message. Thus PeerReview actually increases the probability of a message eventually arriving at the destination. Fig. 2.2 demonstrates a case of message loss: node *j* sends a message *msg* to node *i*, but *msg* is lost because of network problems; node *j* then creates a message *chal(send, msg)* and delivers it to the witnesses of *i*; if *chal(send, msg)* fails to arrive at node *i* then *msg* is eventually lost. If *msg* is lost (no ACK from *i* is received by *j*), then node *j* sends *challenge (send, msg)* to $\omega(i)$. Let *r* denote the number of nodes in $\omega(i)$ that receive *challenge*; therefore, *msg* is eventually lost if and only if all challenges forwarded by witnesses are lost. Let P_0 denote the E2E MLP between any two nodes, and let P_e denote the probability that a message will eventually be lost. Then we have:

$$\begin{aligned} P_e &= P_0 \cdot \sum_{r=0}^{w} \left(\binom{w}{r} \cdot (1-P_0)^r \cdot P_0^{w-r} \cdot P_0^r \right) \\ &= P_0^{w+1} \cdot \sum_{r=0}^{w} \left(\binom{w}{r} \cdot (1-P_0)^r \cdot 1^{w-r} \right) \\ &= P_0^{w+1} (2-P_0)^w \end{aligned} \tag{2.6}$$

2)False positive

Node *i* gives a status indication of another node *j* based on node *j's* evidence set, which will be periodically fetched from node *j*'s witnesses. Once node *i* receives any piece of evidence of node *j*'s accusation, *i* will not 'trust' *j* anymore. However, the witnesses of node *j* might get compromised, and the faulty witnesses will probably attempt to accuse a correct node or ignore a faulty node by issuing false evidence to confuse other nodes. PeerReview is able to defend this threat because all evidences are designed to be unforgeable; thus, as long as there exists one correct witness (we assume this condition always holds, like PeerReview does), the only reason for false positive is that all evidence messages are lost. For PeerReview, $\forall i \in V_H$, after one operation of evidence transfer protocol, we have

$$P_{FP} = P_e^{w\cdot(1-\phi)} \tag{2.7}$$

The evidence transfer protocol will execute multiple rounds in the system life cycle. However, each round is independent. If we capture a snapshot of the whole system, all status indications only depend on the latest evidence transfer operation. The real problem is that φ will be growing, making $\lim_{\varphi\to 1} P_{FP} = \lim_{\varphi\to 1} P_e^{w(1-\varphi)} = 1$, which means that, if all witnesses of *j* become faulty, *i* will keep 'trusting' *j* forever.

3)False negative

False negative includes *X:C* and *S:C*. The cause of *X:C* is twofold: 1) if $i \in \omega(j)$, it might suffer *X:C* because of message loss during audit; 2) if $i \notin \omega(j)$, it will suffer *X:C* because some nodes in $\omega(j)$ have already suffered *X:C* and *i* received faulty evidence from these nodes. For cause 2), assume that there are *r* correct witnesses in $\omega(j)$ that suffer *X:C*, and then these *r* witnesses would generate *r* faulty evidence messages and spread them. If node *i* receives any of

these *r* faulty evidence messages, *i* will suffer *X:C*. For convenience of derivation, we assume that for each auditing, a log segment will be sliced into $\bar{\mu}$ small pieces and loaded into $\bar{\mu}$ messages. The probability of false negative is given below:

$$P_{FN} = \Pr(X:C) + \Pr(S:C) \tag{2.8}$$

in which

$$\Pr(X:C) = \frac{w}{N} \cdot \left(1-\left(1-P_e\right)^{\bar{\mu}}\right) + \left(1-\frac{w}{N}\right) \cdot \sum_{r=0}^{w(1-\varphi)} \left(\binom{w(1-\varphi)}{r} \cdot \left(1-\left(1-P_e\right)^{\bar{\mu}}\right)^r \cdot \left(1-P_e^r\right) \right) \tag{2.9}$$

$$\Pr(S:C) = P_e + (1-P_e) \cdot P_0 \tag{2.10}$$

A proof sketch of (2.9) is given as follows: Consider a correct node *x*. There are two cases that node *x* will be exposed by another correct node *y*. Case 1): node *y* is one of node *x*'s witness (with probability *w/N*), and at least 1 of the $\bar{\mu}$ messages are eventually lost (with probability $(1-\left(1-P_e\right)^{\bar{\mu}})$). Case 2): node *y* is not node *x*'s witness (with probability (*1-w/N*)); if there are *r* witnesses of node *x* that have already suffer *X:C*, they will pass the fault evidence (*r* in total) to node *y*. Therefore, node *y* will suffer *X:C* if and only if at least one evidence of *r* arrive at node *y*.

Combining equations (2.5) - (2.10), we obtain $\bar{A}(Q)$ for PeerReview.

Theorem 2.1: *In the PeerReview context, if a message can eventually be lost during the system lifetime, the entire system will ultimately become non-accountable at all*.

Proof: In a PeerReview-enabled system, eventual message loss leads to incorrect indications, i.e., errors. If more and more messages are eventually lost, more errors will occur and accumulate; in the end, the percentage of correct indications will drop to zero. In other words, P-Accountability reaches 0. Therefore, the system becomes non-accountable at all. □

2.6 Accountable Wireless Multi-hop Networks

PeerReview works in a context with E2E communication. The idea of PeerReview is also applicable to wireless multi-hop networks. In a typical wireless multi-hop network, each host serves as both a host and a relay node. Based on the original PeerReview primitives, however, a faulty node will not be detected when it is acting as a relay. To this end, we propose *message tracing protocol* and add it to the original PeerReview protocol set to hold a faulty relay node accountable.

Wireless channels suffer many network dynamics, such as traffic loss/delay, congestion, and interference. A message might be lost in any hop before reaching the destination. For example, a frame at IEEE 802.11 networks will be discarded after seven (i.e., the default value) tries (transmissions/re-transmissions). Furthermore, many nodes may be up and down at any moment, and wireless connections may be very bad. Therefore, messages may eventually be lost. We keep using message loss as an impact factor to analyze P-Accountability for TPR.

2.6.1 TPR Context

The features of wireless multi-hop networks pose a new context that differs from PeerReview-enabled systems in the following aspects:

1. Each node plays three roles: source, destination, and relay.
2. Suppose that nodes S and D are two end nodes, and a path between nodes S and D is defined as $P_{S,D}$ = <S, $R_1, R_2, ..., R_l$, D>, where $R_1, R_2, ..., R_l$ are all relay nodes. We let $P_{S,D}(m_x)$ represent the path that a specific message m_x flows through. Obviously, a message sent from node S to node D will be delivered through ($l+1$) hops.

3. E2E ACK and Hop ACK: PeerReview has defined the E2E ACK. In multi-hop networks, we enable the Hop ACK, which is for the acknowledgement within hops. When a relay node, R_k, forwards a message to the next hop, it expects an ACK from its successor, R_{k+1}; if an ACK is still not received even after q - 1 retransmissions, node R_k gives up and performs further actions.

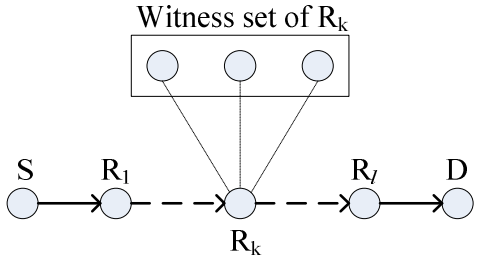


Fig. 2.3 Message delivery in multi-hop networks

2.6.2 Problem Description

New vulnerabilities will arise in a PeerReview-enabled wireless multi-hop network. Fig. 2.3 depicts a general message delivery procedure with nodes S and D being source and destination, respectively. If we consider path $P_{S,D}$ = <S, $R_1, R_2, ..., R_l$, D>. A message, m_x, from node S is able to reach node D if and only if 1) all relay nodes in $P_{S,D}$ are correct and 2) m_x is not lost in the (l+1) transmission hops. If a relay node R_k is faulty, it is possible that:

1. R_k ignores all messages from source *S*, which causes node *D* to never hear from node *S*; thus, the one-way communication between nodes *S* and *D* is totally blocked for the current session.
2. R_k filters traffic from source node *S* either strategically or randomly, making it impossible for node *D* to receive a complete data copy from node *S*.
3. R_k re-sends the same message multiple times on behalf of source node *S*, which might lead to a DoS or replay attack on node *D*.

4. R_k forwards messages from node *S* to other nodes which are not in the path of $P_{S,D}$. As a result, node *D* might lose its connection with node S.

When R_k faithfully maintains its log file, PeerReview is able to deal with these threats since R_k will be exposed eventually by its witnesses. PeerReview then instructs other nodes to expel R_k from their trust lists. However, if R_k is smart enough not to leave any evidence in its log, then even the witnesses will be unable to determine whether or not R_k relayed messages from node *S*. For example, R_k has received a message, m_x, from node *S*, but this action was not recorded into R_k's log; then R_k discards m_x. Later on, when the witnesses check R_k's log, no log entries relevant to m_x exist. Therefore, R_k can hide itself. However, evidence always exists. At least R_{k-1}, the node preceding R_k within the path $P_{S,D}$, did forward messages to R_k, meaning that there are definitely some log entries relevant to m_x existing in R_{k-1}. Thus, a piece of evidence that exposes R_k can be generated. Therefore, the basic idea of TPR is that: when relay nodes are involved, in order to expose a potentially faulty node, it is not enough to just replay the logs of a single node; instead, a cooperative inspection approach is needed to generate the evidence.

We propose Traceable PeerReview (TPR), which ensures 1) regarding a message, m_x, sent from source *S* to destination *D*, TPR is capable of tracing m_x through $P_{S,D}(m_x)$ and identifying the location where m_x is stopped; 2) TPR is able to generate verifiable evidence to expose the first faulty node that m_x encounters within $P_{S,D}(m_x)$.

2.6.3 Traceable PeerReview

1)Changes to the PeerReview protocol set

For the Tamper-evident logs (section 4.4 in [1]), the log entry of each message needs to be extended. The content field should include four addresses: Source ID, Destination ID, last hop node ID, and next hop node ID.

2)Message Tracing Protocol

For source *S* and destination *D*, if *S* does not receive the E2E ACK of a certain message, m_x, from *D*, *S* will trigger the challenge/response protocol (section 4.8 in [1]) and begin to suspect node *D* until the challenge is answered. If node *D* is a correct node, the reason that node *S* incorrectly suspects node *D* is twofold: 1) either m_x or ACK (m_x) is lost in transmission; 2) there exist faulty relay nodes in the path $P_{S,D}(m_x)$.

The Message Tracing procedure is described as follows:

- **Step 1**: If source *S* does not receive the E2E ACK of m_x, then *S* starts to trace m_x through path $P_{S,D}(m_x)$. First, node *S* generates a message *TracingTag* $\sigma_S(seq_x)$, which is a signed statement of m_x's sequence number by *S*, and then delivers it to the next hop, R_1, which is supposed to relay m_x. From this moment on, node *S* becomes a temporal checker of R_1. After R_1 receives the TracingTag, it validates the tag message with node *S*'s public key. If the tag is invalid, R_1 discards it; otherwise, R_1 finds all entries related to seq_x in the log and returns them to node *S*, which is able to check the correctness of R_1. If R_1 is validated within node S, node *S* will send a *Pass* message to R_1, and R_1 formally joins path $P_{S,D}(m_x)$. After this checking round, R_1 generates a new TracingTag, $\sigma_{R_1}(seq_x)$, and sends it to the next hop.
- **Step 2**: As Fig. 2.4 shows, when R_i receives $M_a = \sigma_{R_{i-1}}(seq_x)$ from R_{i-1}, it sends all entries relevant to m_x back to R_{i-1}. R_{i-1} checks the following about m_x in R_i: the source ID, the destination ID, the last hop node ID, and the next hop ID. If all

of these identifiers are correct, then R_i is verified to have forwarded m_x correctly. Otherwise, R_i is proved to be a faulty node. The evidence consists of the entries that R_i sent to R_{i-1}. If R_i is node D, then the message tracing of m_x is ended. Since the log is tamper-evident, any malicious modification to the log will be detected by the witnesses.

- **Step 3**: If there is no faulty node in $P_{S,D}(m_x)$ and D did receive m_x, message tracing of ACK(m_x) starting from node D begins, since we assume that $P_{D,S}(m_x)$ is not necessarily equal to $P_{S,D}(m_x)$.
- **Step 4**: After the witnesses receive the evidence from the temporal checker, they will verify its correctness. Then, the evidence piece will be delivered to other nodes via the evidence transfer protocol (section 4.9 in [1]).

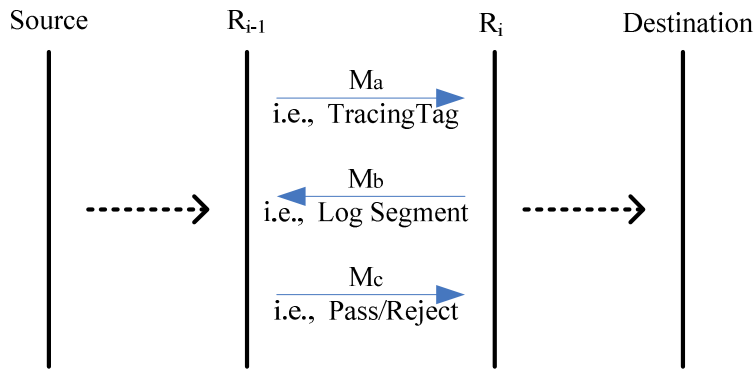


Fig. 2.4 Tracing in path $P_{S,D}(m_x)$

2.6.4 Traceable PeerReview Analysis

Let P_h denote the message loss probability in each hop, and let P_C denote the probability that the message tracing process can be accomplished, meaning that no message is eventually lost during message tracing so either a faulty relay node can be identified or the hop where the

original message is lost can be found. Let $\tau_{i,j}$ denote the hop count between nodes *i* and *j*. Assume that the first faulty node in the tracing path is R_k. We have

$$P_C = \left(1 - P_h^q\right)^{3 \cdot \tau_{S,k}} \tag{2.10}$$

Proof of (2.10): The tracing mission will be successfully accomplished if and only if all relay nodes are reachable, which means messages during tracing need to arrive at the next hop relay node. Each hop checking needs to deliver 3 types of tracing messages (i.e., TracingTag, log segment, and pass/reject), and each message can be re-transmitted (*q-1*) times. Hence, the probability that the one-hop checking is finally successful is $(1 - P_h^q)^3$. Based on the location of R_k, there are two cases:

- Case 1: $R_k \in P_{S,D}(m_x)$, i.e., R_k lies in the path before m_x reaches D. Evidently, the tracing will be accomplished if and only if all hop checking within the $\tau_{s,k}$ hops succeeds. The probability for this case is $\left(1 - P_h^q\right)^{3 \cdot \tau_{S,k}}$.
- Case 2: $R_k \in P_{D,S}(ACK(m_x))$, i.e., R_k lies in the path of ACK(m_x) from D to S. Similarly, we have $P_C = \left(1 - P_h^q\right)^{3 \cdot (\tau_{S,D} + \tau_{D,k})} = \left(1 - P_h^q\right)^{3 \cdot \tau_{S,k}}$.

Therefore, equation (2.10) is proven. □

Theorem 1: *The message complexity of one round message tracing is* $O(\tau_{S,D} + \tau_{D,S})$, *which is proportional to the hop count of the tracing path*.

Proof: Suppose hop level ACK retransmission is enabled and each message can be retransmitted (*q-1)* times before the sender gives up. Based on the mechanism of packet tracing, a node R_i has 3 messages exchanged with its successor R_{i+1} so as to finish the one hop checking. Taking packet loss and retransmission into account, the upper bound of the message count is *3q*

within one hop. In the entire tracing procedure, the maximum number of hops to be checked is $\tau_{S,D}+\tau_{D,S}-1$; thus, the message overhead is $3q\cdot(\tau_{S,D}+\tau_{D,S}+1)$. The complexity for one round message tracing is $O(\tau_{S,D}+\tau_{D,S})$ □

2.6.5 P-Accountability on TPR

Since events *T:C*, *X:C*, and *S:C* are mutually exclusive, we have Pr(*T:C*) = 1- Pr(*X:C*) - Pr(*S:C*). Similarly, Pr*(X:F)* and Pr(*S:I*) can be computed. Therefore, in the TPR case, we have P-Accountability $\bar{A}(Q)$ defined as follows: For any node $v\in V$,

$$\begin{aligned}&\bar{A}(Q)=\Pr(\text{node } v \text{ makes correct indications})\\&=\Pr(T:C)+\Pr(X:F)+\Pr(S:I)\\&=1-\big(\Pr(X:C)+\Pr(S:C)\big)+1-\big(\Pr(T:F)+\Pr(S:F)\big)+1-\big(\Pr(T:I)+\Pr(X:I)\big)\\&=3-\Pr(E:C)-\Pr(S:C)-\Pr(T:F)-\Pr(S:F)-\Pr(T:I)-\Pr(X:I)\end{aligned} \tag{2.11}$$

1)Eventual E2E message loss

For any two end nodes *i* and *j*, the E2E message loss probability is determined via

$$P_{t(i,j)}=1-(1-P_h^q)^{\tau_{i,j}} \tag{2.12}$$

For any two end nodes *i* and *j*, the *eventual* E2E message (sent from *i* to *j*) loss probability is

$$P_{E(i,j)}=P_{t(i,j)}\cdot\left(\sum_{k=0}^{w\cdot(1-\varphi)}\left(\binom{w\cdot(1-\varphi)}{k}\cdot\prod_{r=1}^{k}\left(1-P_{t(i,r)}\right)\cdot\prod_{l=1}^{w\cdot(1-\varphi)-k}\left(P_{t(i,l)}\right)\cdot\prod_{r=1}^{k}\left(P_{t(r,j)}\right)\right)\right) \tag{2.13}$$

Equation (2.13) is the multi-hop version of equation (2.6). The basic idea of calculating $P_{E(i,j)}$ is that if a regular message m_x is lost (with probability $P_{t(i,j)}$), node *i* will send challenges to its witnesses; the message m_x will eventually be lost if and only if all challenges forwarded by witnesses are lost.

2)Error analysis

The six error types are consistent with the ones discussed in section 2.5. In addition, we differentiate normal faulty (NF) nodes (denoted by V_{NF}) from relay faulty (RF) nodes (denoted by V_{RF}). Let $V_{NF} \cup V_{RF} = V_F$ and $V_{NF} \cap V_{RF} = \varnothing$, and let p_{nf} and p_{rf} denote the probabilities that a node is a normal faulty node and a relay faulty node, respectively. Again, given a correct node *i* and another node *j*, we can calculate the probabilities for errors as follows:

- *Error* $\left(u_{j,i} = X\right) : \left(o_j = C\right) = X : C$. TPR will not add other causes of error $X : C$ because, in message tracing, a correct node can always provide the verifiable log piece M_b to its temporal checker. We show the multi-hop version of equation (2.9).

$$\begin{aligned} &\Pr(X:C) = \frac{w}{|V|} \cdot \left(1 - \left(1 - P_{E(i,j)}\right)^{\bar{\mu}}\right) + \\ &\left(1 - \frac{w}{|V|}\right) \cdot \sum_{r=0}^{w(1-\varphi)} \left(\binom{w(1-\varphi)}{r} \cdot \left(1 - \left(1 - P_{E(i,j)}\right)^{\bar{\mu}}\right)^r \cdot \left(1 - P_{E(i,j)}^r\right)\right) \end{aligned} \tag{2.14}$$

- *Error S:C and S:F*
  - Case C1: node *i* is the source node, and node *j* is the destination node. Node *i* suffers *S:C* or *S:F* if and only if 1) a message from *i* to *j* is eventually lost or 2) *j* has received the message but the ACK message is eventually lost. Therefore, we have

$$\Pr(C1) = P_{E(i,j)} + (1 - P_{E(i,j)}) \cdot P_{t(i,j)} \tag{2.15}$$

  - Case C2: nodes *i* and *j* are two adjacent nodes in the path. Assume that node *i* is the precedent of *j*, which could be either correct or faulty.

$$\Pr(C2) = P_h^{2q} \cdot P_{E(i,j)} \tag{2.16}$$

Since C1 and C2 are mutually exclusive, we have

$$\Pr(S:C)+\Pr(S:F)=\Pr(C1)+\Pr(C2) \tag{2.17}$$

- *Error T:F and T:I.* TPR adds one more cause of *T:F*: if a relay faulty node has been exposed by its temporal checker *t*, but the evidence message is lost, and then other nodes will keep trusting this node. Therefore, we have

$$\Pr(T:F)+\Pr(T:I)=\frac{w}{|V_H|}\cdot p_{rf}\cdot P_{E(t,i)}+\left(1-\frac{w}{|V_H|}\right)\cdot p_{nf}\cdot\prod_{r=1}^{w\cdot p_c}P_{E(r,i)} \tag{2.18}$$

- *Error X:I.* TPR can also ensure that an ignorant node will not be exposed by mistake. Thus, $\Pr(X:I)=0$

Based on equations (2.11) – (2.18), we obtain $\bar{A}(Q)$ for TPR.

2.7 Evaluation

In this section, we provide both numeric and simulation results for P-Accountability.

2.7.1 Numerical Results

1)PeerReview

$\bar{A}(Q)$ for PeerReview can be numerically plotted in Fig. 2.5. In this figure, P-Accountability is a probability affecting by the E2E message loss probability and the witness size. The reason that a larger witness set increases P-Accountability is that once a message is lost, there will be more witnesses to forward the challenge to the destination.

2)Traceable PeerReview

A successful tracing means that there is no eventual loss of tracing messages in the tracing path. Two factors, i.e., hop re-transmission time (*q-1*), and the average hop count τ_{avg} between the source and the first faulty node, affect a successful tracing. Fig. 2.6 shows that P_C is

positively affected by q and negatively affected by τ_{avg} . Evidently, more re-transmissions imply a lower probability of message loss, while a higher hop count does the opposite.

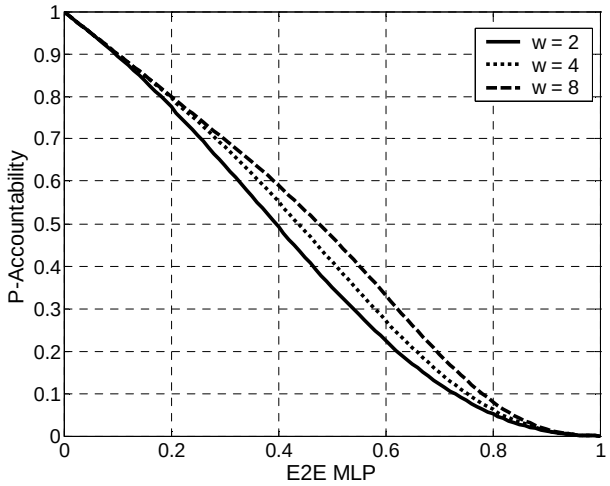


Fig. 2.5 The E2E MLP and P-Accountability for PeerReview when φ = 0.1

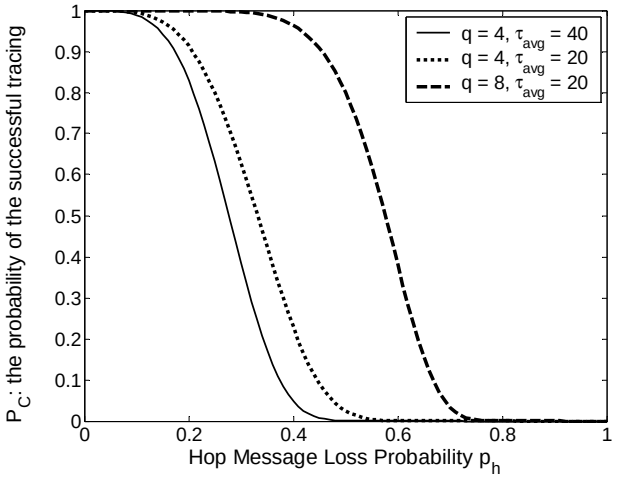


Fig. 2.6 Hop MLP and P_C for TPR (φ = 0.1)

Fig. 2.7 shows how hop message loss probability P_h affects P-Accountability for TPR. Basically, P-Accountability decreases with the rise of P_h. In addition to the witness size, another factor is the average hop count between two end nodes. Both two factors exists because they affect the eventual E2E message loss probability.

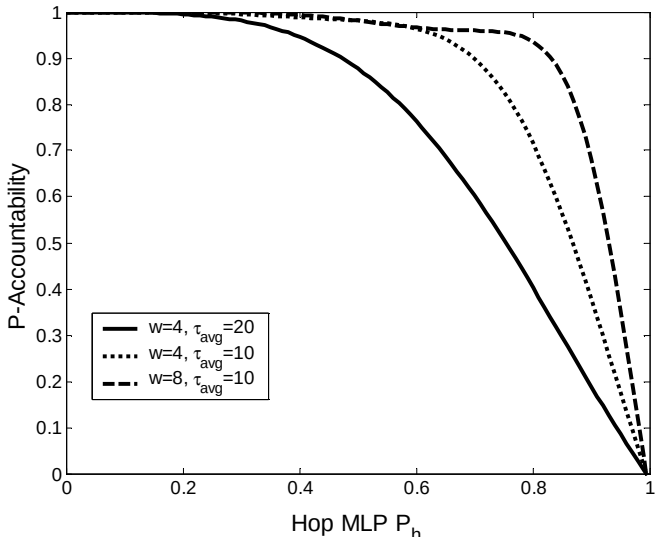


Fig. 2.7 Hop MLP and P-Accountability for TPR (φ = 0.1)

2.7.2 Simulation Results

TPR is an extension of PeerReview, and our simulation program is also extended from the original PeerReview software library [26]. Fig. 2.8 depicts the framework of the TPR-enabled network system. Similar to PeerReview, TPR is also application independent. In other words, multiple programs are able to install TPR as an add-on so that they are accountable for general Byzantine faults. Based on the hierarchical definition of accountability, two hierarchies can be identified in a TPR-enabled multi-hop network system: 1) H_1, the higher hierarchy that contains all nodes, and 2) H_2, the lower hierarchy that is comprised of multiple applications.

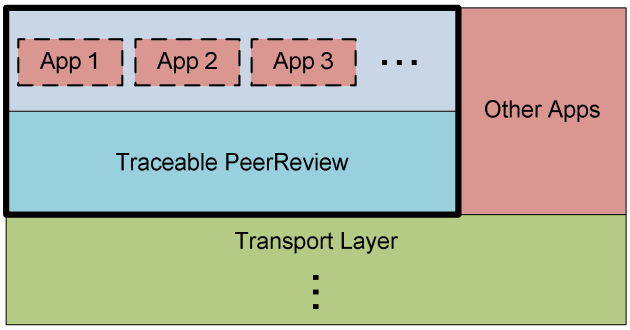


Fig. 2.8 Framework of TPR enabled system

TPR keeps one log for each application, and each node has a key pair as its identity. Applications work independently. Witnesses lie in H_1, implying that all applications of one node share the same witness set. Since a node has multiple TPR-enabled applications running on it, we are able to define an experimental metric (see equation (2.19)) that is consistent with equation (2.3). We define the event space as {the status ('Trust', 'Expose', or 'Suspect') of an application x of node y is indicated by every other correct node}. The entities are nodes (in H_1) and applications (in H_2). Assume that there are M applications running on each node. Thus, the event space is $M \cdot |V| \cdot |V_H|$.

$$A_H(Q) = \frac{\sum_{m \in M} \sum_{v \in V_H} \sum_{k \in V} \left(I(u_{k,v} \text{ is correct}) \right)}{M \cdot |V| \cdot |V_H|} \tag{2.19}$$

where the numerator item denotes the number of correct indications of node v. In this case, a perfect mapping correctly identifies an application (H_2) and not just a node (H_1). Based on the nature of TPR, there is only one scenario in which an indication is correct but not perfect: if a node x is an ignorant node, and therefore other nodes can only be aware that x should be suspected but they do not know whether the applications running on x are still good or not.

1)P-Accountability on Traceable PeerReview

The original PeerReview library sought to ensure that a message will eventually arrive at its destination. In our simulations, each message has a certain probability of being lost in each hop; also, the number of re-transmissions is limited. Thus, a message does have a chance to be lost eventually. Our objective is to determine how message loss affects P-Accountability. We simulate a simple wireless multi-hop network deployed within a square area. Each node has two applications installed. All nodes are homogeneous. The parameters specified in the simulation are listed in Table 2.3.

Table 2.3 Simulation Parameters

Parameter	Value
P_h – hop message loss probability	0.0, 0.1, 0.2 up to 1.0
N – node number in total	[50, 100, 250, 500]
φ – initial faulty nodes rate	[0.0, 0.05, 0.1, 0.2]
w – witness size	[2, 4, 8]

Table 2.4 Error ratios for Traceable PeerReview

Indication \ P_h	0.2	0.4	0.8
correct	73%	55%%	37%
X:C	1.5%	2.3%	1.7%
S:C	12%	19%	28%
T:F	2.4%	3.9%	4.2%
S:F	9%	15%	21%
T:I	2.0%	4.9%	8.1%

Table 2.4 shows the percentage of each indication type in the simulations. We find that, with P_h increasing from 0.2 to 0.8, the accurate indications decrease by 52.6%. Meanwhile, the number of occurrences of each error rises. We also observe that, for the five errors, *X:C*, *T:F*, and *T:I* rarely occur. *X:C* is triggered if and only if messages are lost in the audit protocol; thus, a witness is unable to generate output as expected. Usually, a node has more than one correct witness. Hence a temporal *X:C* could occur, but it would probably jump back to the right indication later. The cause for *T:F* and *T:I* is that evidence messages are lost in the evidence transfer protocol. Additionally, *S:C* and *S:F* do occur quite often since they are readily triggered once a message, or its ACK, is lost. Some of these errors are resolved, but the rest make the system less accountable.

Fig. 2.9 shows the relationship between the hop count message loss probability P_h and P-Accountability $A_H(Q)$. The result is consistent with our numeric result in Fig. 2.7. It shows that 1) P-Accountability declines as P_h rises and that 2) when P_h is given, a larger witness size is beneficial for accountability. A larger witness size would reduce the probability of errors. A challenge will first be forwarded by more witnesses, and the node being challenged must respond to each of them; thus, the response message has a higher chance to reach the other end. Second, more witnesses would reduce the probability that an evidence message is eventually lost in the evidence transfer protocol, leading to the occurrence of *T:F* and *T:I*.

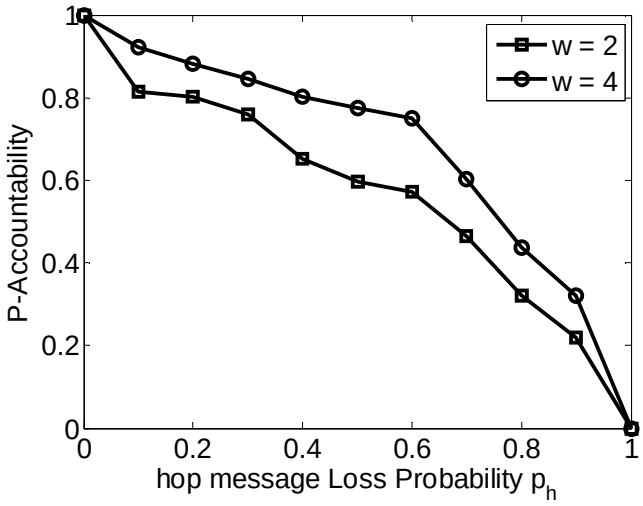


Fig. 2.9 Simulation result - P_h and P-Accountability

In this context, each message is set to be lost with probability P_h in each hop. In addition, we insert some relay faulty nodes into the system. We then compare the PeerReview-enabled system to the TPR-enabled system in terms of P-Accountability. Fig. 2.10 shows that, when P_h is fixed, the latter is more accountability. The main reason is that TPR is able to expose the faulty relay nodes which would not be detected in a PeerReview-enabled system.

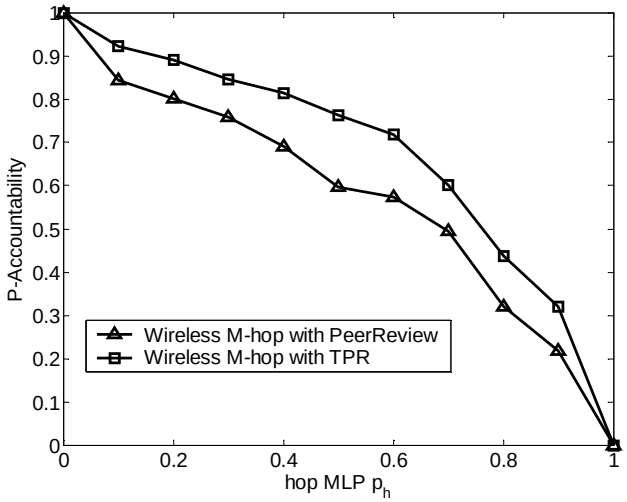


Fig. 2.10 Simulation result - hop MLP and P-Accountability for TPR

2.8 Conclusion

P-Accountability is our attempt to quantify and evaluate accountability for networked systems. To achieve this goal, we have developed a flat model and a hierarchical model. Both models are generic and widely applicable. The flat model suits systems with flat structure, meaning that all concerned entities are homogeneous. The hierarchical model is an advanced version that applies to more complex environments. P-Accountability should be customized before applied to a practical system. To examine our model, we first conduct a case study that applies P-Accountability to PeerReview. In addition, we propose TPR which enables PeerReview to be used in the wireless multi-hop network by holding relay nodes accountable. We conduct extensive simulation to validate our analysis. We show that in the PeerReview case, message delivery plays a crucial role in determining accountability. We also discover that TPR is superior to the original PeerReview version. Both our analysis and simulation results show that P-Accountability makes accountability more flexible, adjustable, and fine-grained for a networked system.

CHAPTER 3

ACCOUNTABLE MAPREDUCE IN CLOUD COMPUTING

3.1 Introduction

MapReduce [32] has been widely used as a powerful tool for data processing. It has efficiently solved a wide range of large-scale computing problems, including distributed grep, distributed sort, web-link graph reversal, web-access log stats, document clustering, etc. Cloud Computing presents a unique opportunity for batch-processing and analyzing terabytes of data that would otherwise take hours to finish [44, 47]. Most cloud providers (e.g., Google, Yahoo!, Facebook, etc.) adopt MapReduce to build their multitenant computing environments. Usually, cloud customers have a large data set to be processed under certain time constraints. For example [40], The New York Times used 100 Amazon EC2 instances and a Hadoop MapReduce [33] application to convert 4TB of raw image TIFF data (stored in S3) into 11 million PDFs in 24 hours at a computation cost of about $240 (not including bandwidth).

In such a computing environment, the cloud customers outsource their data to the cloud, which performs the storing and computing operations required by the customers. Customers must, therefore, fully trust the cloud provider. However, a cloud provider cannot guarantee that every machine in its data center is 100 percent trustworthy. Some machines may become malicious if they are attacked and controlled by hackers; malicious machines will not faithfully carry out the tasks assigned to them. As a result, the processing result is no longer correct and trustworthy. In the New York Times example, malicious nodes may mess up the image

conversion process, leading to a mismatch between the PDFs and the original TIFF images. It is even harder for the New York Times to check if these PDFs are correctly converted because of the tremendous data size.

In this chapter, we propose building an Accountable MapReduce to make the cloud version MapReduce trustworthy. We design a primitive called Accountability Test (A-test), which checks all working machines when a job is undertaken and detects malicious nodes in real time. The A-test is performed by a group of trusted machines, called the Auditor Group (AG). The AG takes advantage of the deterministic property of a user's MapReduce program to replay the tasks executed by working machines. The MapReduce framework makes it possible for an auditor to acquire the input data block and processing results without notifying the working machines. If the replay output does not match the original output, it implies that the worker returned bad results, and the evidence is the combination of the task, input, original output, and replay output.

A challenge of Accountable MapReduce is the reduction of the A-test overhead. In theory, the A-test can guarantee the detection of any misbehavior by fully duplicating each task, and this at least doubles the processing time. To make the A-test more efficient, we give up pursuing 100% accountability; instead, we provide probabilistic guarantee for accountability. Based upon the batch-processing property of MapReduce, the performance of A-test with P-Accountability can be greatly improved by decreasing the degree of accountability by less than 1%. We summarize the contributions of this chapter as follows:

- We propose Accountable MapReduce in the cloud setting to detect malicious machines. Verifiable evidence will be generated to ensure that the malicious nodes cannot deny their misbehavior.

- Instead of pursuing perfect accountability, A-test allows the system to achieve P-Accountability with less overhead and faster processing time.
- We formulate the Optimal Worker and Auditor Assignment (OWAA) problem for Accountable MapReduce. The OWAA problem aims to find the optimal numbers of workers and auditors in order to minimize the total processing time.

3.2 Related Work

The ability to process data-intensive tasks has made MapReduce increasingly important in the distributed computing field. Chu *et al*. [38] applied MapReduce to machine learning on multi-core platforms. He *et al*. [41] implemented Mars, a MapReduce framework, in graphics processors. Papadimitriou *et al*. [43] explored the usage of MapReduce in data mining. Ekanayake *et al*. [42] adopted MapReduce for two kinds of scientific data analyses, including high energy physics data analyses and K-means clustering. Existing work focuses on utilizing MapReduce to solve different domain problems. General security concerns [36, 37, 39] are discussed, as well. However, accountability issues in MapReduce have not been well addressed. Accountable MapReduce is a proposal to address the issue of misbehavior detection for MapReduce in the cloud setting.

Wei *et al*. [35] present SecureMR, a practical service integrity assurance framework for MapReduce. The authors design a decentralized, replication-based integrity verification scheme to ensure the computational integrity of MapReduce in open systems. SecureMR pursues 100 percent integrity for MapReduce, which will slow down the data processing. For some applications, efficiency is more important than perfect integrity. In this case, computational integrity is conceptually equivalent to accountability. Therefore, instead of pursuing 100 percent

accountability, we allow the customers to choose the level of accountability. It turns out that slightly decreasing the expectation of accountability drastically reduces the system overhead and greatly improves the processing speed.

Accountability has been regarded as an important issue in cloud computing. The trust relationship between the cloud provider and cloud customers has been addressed in [45, 46]. The customer places his/her computation tasks and data on remote machines; the cloud provider agrees to run a service without knowing the details [45]. Therefore, accountability is employed to determine whether or not the Service Level Agreement (SLA) is fulfilled. If it is not, evidence should be provided in order to find out the responsible party.

3.3 MapReduce Background

3.3.1 Programming Model

With MapReduce, programmers only need to specify two functions: *Map* and *Reduce*. The Map function parses the input and generates intermediate key/value pairs to be further processed. The Reduce function merges all the intermediate key/value pairs associated with the same (intermediate) key and then generates the final output.

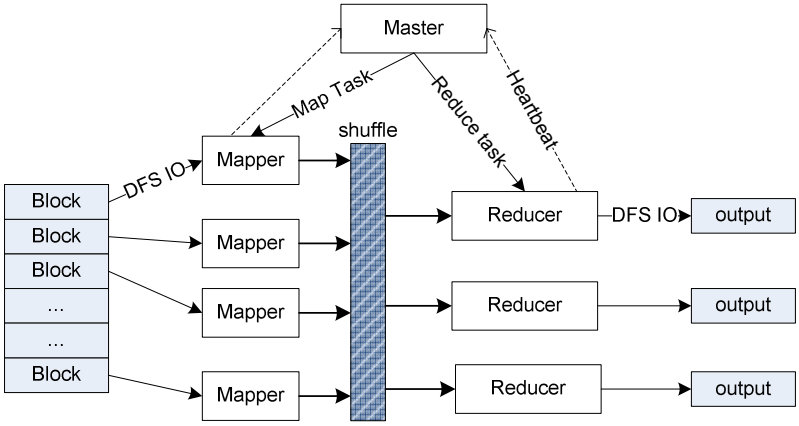


Fig. 3.1 MapReduce Working Flow

There are three main roles: the master, mappers, and reducers. The master is a singular, acting as the coordinator that is mainly responsible for task scheduling, job management, etc. MapReduce is built upon a distributed file system (DFS), which provides distributed storage. Fig. 3.1 shows the execution process of MapReduce. The input data is split into *M* blocks, which will be read by *M* mappers through the DFS I/O. Each mapper will process the data by extracting the key/value pair from the input data set, and then, it will generate the intermediate results that are stored in its local file system. The intermediate result will be sorted according to the keys so that all pairs with the same key will be grouped together (i.e., the shuffle phase). If the memory size is limited, an external sort should be used. The locations of the intermediate results will be sent to the master who notifies the reducers to fetch the intermediate results as their inputs. Reducers then use the Remote Procedure Call (RPC) to read data from mappers to launch the reduce phase. During reduce, the reduce function is applied to the sorted data. Records with the same key will be reduced in some way according to the reduce function. Finally, the output will be written to the DFS, and then returned to the cloud customer.

3.3.2 Fault Tolerance

MapReduce is designed to be fault tolerant because failures happen a lot in a large scale distributed computing environment.

1)Worker failure

The master pings every mapper and reducer periodically. If no response is received for a certain amount of time, the machine is marked as failed. The ongoing task and any tasks completed by this mapper will be re-assigned to another mapper and executed from the very beginning. Completed reduce tasks do not need to be re-executed because their output is stored in the global file system.

2)Master failure

Since the master is a singular, MapReduce will re-start the entire job if the master fails.

3)Byzantine fault tolerance

A Byzantine fault [7] is an arbitrary fault that occurs during the execution of an algorithm in a distributed system. It consists of both omission failures (e.g., crash failures, failing to receive a request, or failing to send a response) and commission failures (e.g., processing a request incorrectly, corrupting local state, and/or sending an incorrect or inconsistent response to a request).

The MapReduce framework can suffer both omission failures and commission failures. The omission failures can be properly solved by the MapReduce built-in fault tolerance mechanisms. However, the commission failure is not addressed in the original version.

3.4 Problem Statement

3.4.1 Attack Model

Fault-tolerance deals with node failures, such as a worker not responding to the master or a worker machine being totally crashed. To address node failures, the master usually re-executes the failed Map or Reduce task in another machine. However, fault-tolerance is unable to detect a malicious node intending to alter the Map/Reduce function or tamper with processing results, and return inaccurate results. For example, Wordcount is a typical MapReduce application. Its job is to count the occurrence of each word in a text file. If there are malicious working machines in the system, the output file, which contains the word count for every word appears in the text, is inaccurate. Consider the Wordcount example in Fig. 3.2. Assume that the system is free of malicious nodes. There are three mappers, each of which maps a line of the text. After mapping,

we obtain the map output as the intermediate results, which will be shuffled (sorted by key) and read by reducers (five of them, in this case). The final output is produced by reducers. If every node faithfully executes its task, the final output will be accurate. Otherwise, the results may not be trustworthy. There are multiple ways for a mapper to alter the output: it can 1) filter some keys, 2) create keys that do not exist in the input file, 3) modify the value on purpose, etc. A malicious reducer can misbehave similarly.

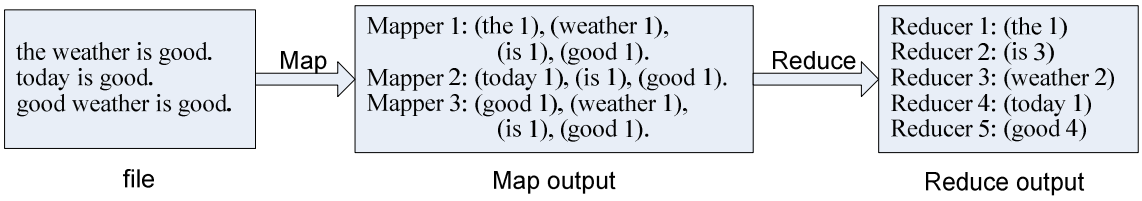


Fig. 3.2 A Wordcount example of MapReduce

3.5 Accountable MapReduce

3.5.1 Design Principles

The key function of Accountable MapReduce is detecting malicious nodes that produce inaccurate results. We now present the principles of our design:

- The accountability mechanism should be silent so that malicious nodes are unaware what is happening. We assume that machines can be fully controlled by attackers, who may be smart enough to take countermeasures to cover their malicious activities.
- The overhead introduced by the accountability mechanism should be minimized.
- When a malicious node is caught, the system can provide verifiable.

3.5.2 Assumptions

The design of Accountable MapReduce is based upon the following assumptions:

- The AG is a trustworthy domain, meaning that the machines in the AG are free of any malicious actions.
- A worker cannot be reclaimed until the entire job is completed. When the customer confirms that the job is done, all machines will be reclaimed by the cloud.
- A malicious node randomly manipulates the processing result. This means that the faulty parts of the processing result also distribute randomly throughout the entire result.
- All input data, intermediate results, and output data will not be removed until the entire job is finished.
- Data from a cloud customer is correct.

3.5.3 Accountable MapReduce Design

1)Correctness checking scheme

PeerReview [1] provides accountability for distributed systems. It assumes that every node in the system is a deterministic state machine. PeerReview employs two critical schemes as building blocks, including tamper-evident logging and witnessing. A tamper-evident log is implemented by a hash chain, which guarantees that any modification to the log will be detected so that a node has to record its behavior faithfully. A witness, which is also a regular node, is able to check the correctness of other nodes that it is witnessing by replaying the log files kept in each node. As a result, malicious nodes will eventually be detected and exposed to all other correct nodes. PeerReview is applicable to most distributed applications. However, it is not suitable for MapReduce. The major concern is overhead. First, the input of a large task may be at the TB level or even PB level (even though there are thousands of workers, each split task also

has a large workload), and the output depends upon the input. It is not acceptable to log all input and output events for correctness checking. Second, a node has to upload its log segment to multiple witnesses for verification, which is extremely bandwidth-consuming.

For Accountable MapReduce, the idea of correctness checking is as follows. Assume that an auditor is a trustworthy node, and that both the worker and the auditor are regarded as deterministic state machines. If the inputs are the same, the outputs should be the same as well. After comparing the worker's output to the auditor's output record by record, the system is able to determine whether the worker is malicious or not. The evidence is the combination of the worker's input and output, and the auditor's output. Additionally, it is verifiable to any other auditors.

2)Auditor Group (AG)

The *AG* performs the Accountability Test (A-test) to detect malicious nodes. Normally, as shown in Fig. 3.3, cloud resources will be split into multiple slices, each of which is rented by a customer. A slice is a group of working machines (either virtual machines or physical machines) assigned to a customer. We maintain an AG manager for the entire cloud and one AG for each slice that runs MapReduce. The purpose for associating each slice with one AG is to conserve the privacy and independence for customers.

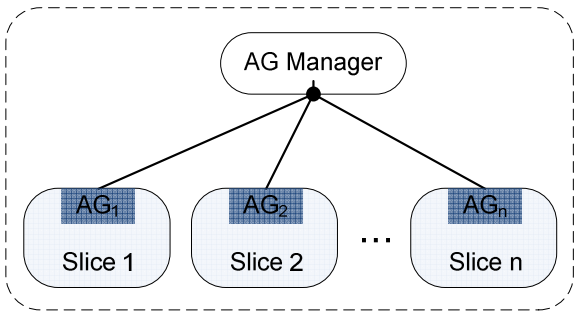


Fig. 3.3 Auditor Group in cloud platform

The AG Manager is designed for AG creation, management, and disposal. After the AG manager is informed of the customer's data size, timing, and other requirements, it will determine the AG size and then create an AG for the slice.

Each AG is internally structured as a cluster, as shown in Fig. 3.4. The head node is the Group Head (GH), and the member node is the Group Member (GM). The GH randomly picks workers as test targets. The master provides the GH with all the information needed for an A-test. The GH then assigns A-test tasks to the available GMs, which are the actual machines that accomplish A-test.

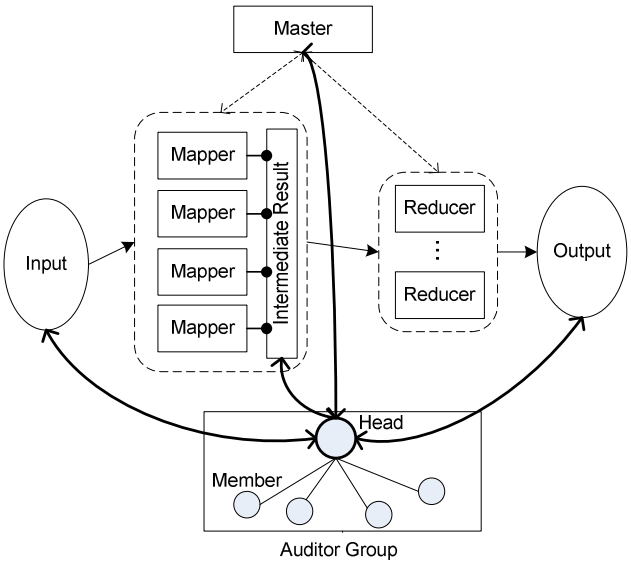


Fig. 3.4 Accountability Test

3)Accountability Test

A-Test is built upon the correctness checking scheme we adopt in this chapter. The AG consists of a set of trustworthy machines dedicated to performing the A-test as shown in Fig. 3.4. The working flow of A-test is described as follows:

1. A-test is started when the job starts.

2. A group of idle auditors will be chosen as the AG for a certain slice. The AG forms a cluster, and only the GH interacts with the master. The GH is thus able to request job information from the master. Therefore, the GH knows where to find 1) the input data and output (i.e., the intermediate data before it is shuffled and sorted) of each mapper; 2) the input (i.e., the intermediate data after it is shuffled and sorted) and output (i.e., the final result) of each reducer.
3. Once the job begins, the GH will receive test tasks from the master, which will be notified once a worker finishes its task. Based upon the processing sequence of map/reduce, the mappers will finish first, and then, the reduce process starts. Therefore, in the initial period, mappers will be tested, and then reducers will be tested after the mappers.
4. After the GH receives a test task to check a worker, it finds an available GM to carry out the test. Each A-test is executed as follows:
    a. The GM will find corresponding input and output based upon the task type (i.e., map/reduce).
    b. The GM will process the input data again and compare its output with the original one to check for consistency. If there is any inconsistency, it indicates that the worker being tested is malicious.
    c. The GM reports the test results to the GH, which will report back to the master.
5. If a worker is detected as malicious, the task it was assigned will be resubmitted to another worker so that the job continues.

4)A-test with P-Accountability

If the auditor processes the entire input for a worker, then the system can definitely determine whether the worker is malicious or not. This implies that the task assigned to the worker is fully replicated. For the whole system, the entire job will be executed twice, once by regular workers and once by the auditors. However, it is not surprising that it takes longer to process the data set twice due the large volume and high overhead. Therefore, instead of pursuing perfect accountability, we combine A-test with P-Accountability, which trades the degree of accountability for efficiency.

Definition 3.1: *P-Accountability in cloud MapReduce* is defined as the probability that a malicious worker will be detected when it tampers with the processing result.

Let P_A denote P-Accountability, and let *w* denote the total number of records in an input file, which can be either a raw data block for a map operation or a partition of intermediate results for a reduce operation. Assume that for any one record, the probability for a worker being malicious is p_m. Let variable *x* denotes the number of records to be checked if we want to achieve P_A. If $P_A = 1$, *x* is equal to *w*, meaning that the entire input file is checked; thus, $1-(1-p_m)^x \geq P_A$. We have $x \geq \left\lceil \log_{(1-p_m)}^{(1-P_A)} \right\rceil$. It shows that if $P_A = 0.9999$ and $p_m = 0.01$, we have *x* = 917, which means that only 917 records need to be checked. We observe that *x* will not be affected by input data size, and only p_m and P_A will be related to *x*. Fig. 3.5 shows how *x* changes when P_A increases from 0 to 1. In fact, when P_A increases, auditors need to check more records to achieve a higher P_A. We also observe that a smaller p_m indicates that there are more records that an auditor needs to check because the malicious node has smaller chance to misbehave.

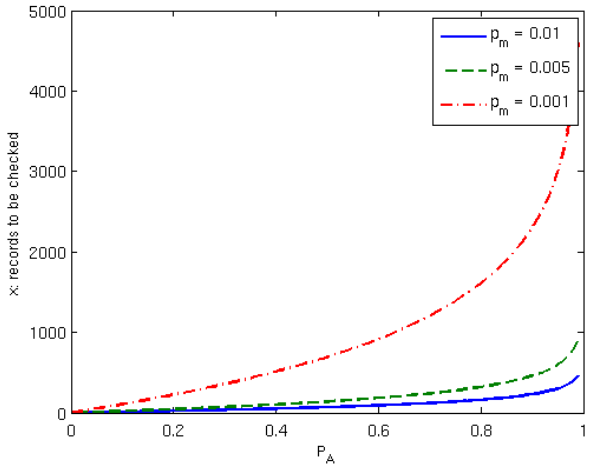


Fig. 3.5 P_A vs. x

The features of A-test are described as follows:

- It is practical to implement the A-test, since it takes advantage of the existing properties of MapReduce. One important task for A-test is to acquire the input and output data of mappers/reducers, and the master has already kept this information. Therefore, no direct contact between auditors and workers is needed. In addition, workers do not know that they are being tested, and it is hard to take countermeasures to hide bad behavior.
- It is an online test, thus the malicious nodes will be detected as early as possible. The design of A-test ensures that a worker will be tested once it finishes.
- With P-Accountability, A-test will be very efficient since a relatively lower P-Accountability will significantly reduce the amount of records to be checked.

One limitation is that false positives may occur if P-Accountability is less than 1. But when the parameters are adjusted, false positives rarely happen.

3.6 Implementation of Accountable MapReduce

3.6.1 Implementation of the Master

The master is the coordinator that holds all information necessary to conduct the A-test. The master has to maintain the following lists: mappers (we denote a list of mappers as *M*), reducers (i.e., set *R*), input set (i.e., *B*), intermediate result set (i.e., *H*), and output set (i.e., *O*). Also, the master node is aware of every input/output relationship in the system. Therefore, a four-tuple set will be kept in order to respond to the requests from the GH: <type, ID, input, output>, where type is the worker type (i.e., mapper or reduer), ID is the worker's identity, and input and output depend on the worker type. Table 3.1 shows the input and output in MapReduce. Let the map output $\{h_{i,1}, h_{i,2}, ..., h_{i,k}\}$ be the intermediate result (produced by worker m_i) before it is shuffled and sorted; let the reduce input $\{h_{1,i}, h_{2,i}, ..., h_{n,i}\}$ be the intermediate result (will be processed by reducer r_i) after it is shuffled and sorted. These sets are not complete in the beginning. Thus the master will maintain them while the job is running.

Table 3.1 Input and Output in MapReduce

	Input	worker	Output
Map	b_i	m_i	$\{h_{i,1}, h_{i,2}, ..., h_{i,k}\}$
Reduce	$\{h_{1,i}, h_{2,i}, ..., h_{n,i}\}$	r_i	o_i

3.6.2 Implementation of the Auditor Group

Fig. 3.6 shows the message flow during the A-test. Based upon the MapReduce primitives, the master will be notified whenever a worker is done with its task. To perform the A-test, the master notifies the GH through Message 1. There are two types of Message 1:

- **Case 1**: if the worker is a mapper, say, m_i, then Message 1 = (MAP, m_i, b_i, $\{h_{i,1}, h_{i,2}, ..., h_{i,k}\}$), which includes all information about the input and output of m_i. Message 2 is an assignment message of the A-test. The GH will randomly pick a worker that has not yet been tested to generate a test assignment. Suppose m_j is picked as the test object, then Message 2 = (MAP, m_j, b_j, $\{h_{j,1}, h_{j,2}, ..., h_{j,k}\}$). To accomplish the test, the GM reads input block b_j from the DFS (i.e., action 3-a) intermediate result $\{h_{j,1}, h_{j,2}, ..., h_{j,k}\}$ from mapper m_j (i.e., action 3-b). The GM is then able to perform the A-test.
- **Case 2**: if the worker is a reducer r_i, then Message 1 is defined as (REDUCE, r_i, $\{h_{1,i}, h_{2,i}, ..., h_{n,i}\}$, o_i). Suppose r_j is picked as the test object, then Message 2 = (REDUCE, r_j, $\{h_{1,j}, h_{2,j}, ..., h_{n,j}\}$, o_j). Action 3-b means reading $\{h_{1,j}, h_{2,j}, ..., h_{n,j}\}$ from the local disk of every mapper, and action 3-c means reading o_j from the DFS. The GM is then ready to perform the test.

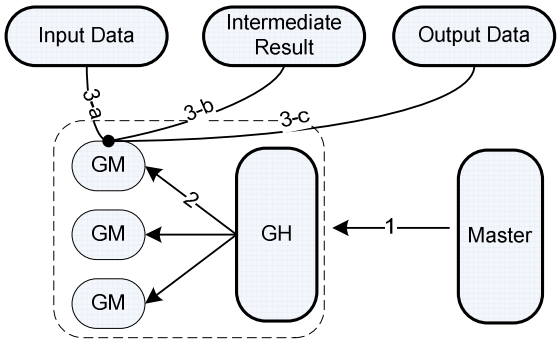


Fig. 3.6 Communication for the AG and Other MapReduce Entities

1)Auditor Group Head (GH)

The GH maintains a list, *L*, of test tasks; *L* is a FIFO queue and will be updated in real time. When a GM is available, the GH will assign a new test task (i.e., the head of *L*) to it. When

the GH is notified that worker w_i is done, the GH produces a test task for w_i immediately so that each worker will be tested at least once. The GH collects the test results from the GM and reports to the master if any misbehavior is detected.

2)Auditor Group Member (GM)

The intermediate result of MapReduce is stored in the mappers' local disks. According to our design, GMs need to fetch the intermediate result from mappers. However, mappers are not able to distinguish testers from reducers, which also need to fetch data from them. Therefore, mappers do not know when they are tested. Table 3.2 shows the A-test algorithm.

Table 3.2 A-test

```
Algorithm: A-test
Require: p_m , P_A , task l ∈ L
x ← ⌈ log_(1−p_m)^(1−P_A) ⌉ // number of records to be checked
If l.type = MAP
    For record i in l.input and i < x
        tmp ← map(l.input[i])
        If tmp is not equal to l.output[i]
            Report inconsistent MAP
If l.type = REDUCE
    while(l.input[x].key =l.input[++x].key);
    for key k in l.input
        tmp ← reduce(k, list(v))
        if tmp is not equal to l.output(k)
        report inconsistent REDUCE
```

3.7 Optimization of Accountable MapReduce

The Accountable MapReduce introduces auditors to the system. There is no doubt that the existence of auditors will introduce extra overhead that slows down the computation process. The remaining question is how to determine the sizes of workers and auditors so that the total

processing time is minimized. We discuss two cases based upon whether auditors are a part of the customer working group. In this section, we formulate the Optimal Worker and Auditor Assignment (OWAA) problem, which takes aim at minimizing the total processing time with the given MapReduce parameter set. Notations of the OWAA problem are given in Table 3.3.

Table 3.3 Notations

T	Total processing time of a job
T_m	Map phase time. It covers the entire Map phase and A-test for Map phase.
T_s	Shuffle time. Since $T_s \propto m$, and $T_s \propto w_2$, we have function $T_s = f_s(m, w_2)$
T_r	Reduce phase time. It covers the entire Reduce phase and A-test for Reduce phase.
T_0	Total processing time that required by a customer. For example, the customer may need the job accomplished in 20 hours, i.e., $T_0 = 20$ hr.
h	Number of records to be checked during A-test. Based on section V, we have $h = \lceil \log_{(1-p_m)}^{(1-P_A)} \rceil$. The larger h is, the longer it needs for A-test.
w_1	The initial Job workload (i.e., data set volume before Map function)
w_2	The intermediate job workload (i.e., data set volume after Map function). Let w_2 be a function of w_1, so we have $w_2 = f_w(w_1)$.
a	Number of auditors
n	Number of workers in a customer's working group; n is constant.
m	Number of mappers
r	Number of reducers
b	Size of each data block; default value is 64 MB
α	Process time for one individual map task. α primarily depends on b, the host application, data set type, machine computation capability (e.g., CPU number, RAM size, etc.). In this case, we only keep factor b, i.e., $\alpha = f_\alpha(b)$
β	Process time for one individual reduce task. Similar to α, we let $\beta = f_\beta(b)$
$\bar{\alpha}$	Process time for one individual A-test for a map task. Since $\bar{\alpha} \propto w \propto h$, we let $\bar{\alpha}$ be a function of h, i.e., $\bar{\alpha} = f_{\bar{\alpha}}(h)$.
$\bar{\beta}$	Process time for one individual A-test for a reduce task. Since $\bar{\beta} \propto w \propto h$, we let $\bar{\beta}$ be a function of h, i.e., $\bar{\beta} = f_{\bar{\beta}}(h)$.

3.7.1 Formulation of Optimal Worker and Auditor Assignment (OWAA) Problem

We have the following assumptions for the OWAA problem:

- The reduced workload for each Reducer can be equally partitioned.
- We assume that there is no hardware difference between workers and auditors.
- The size of workload is the only factor that determines the process time for Map/Reduce/A-test.

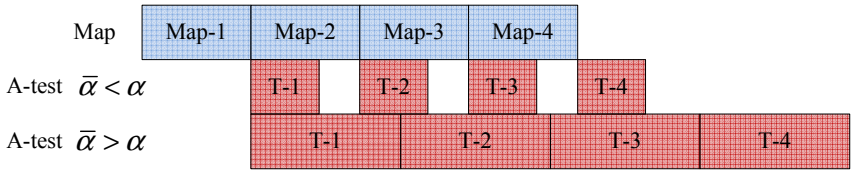


Fig. 3.7 Pipelining the A-test and Map

Fig. 3.7 shows the pipelining of the A-test and Map. We can observe that normally each mapper will process multiple map tasks. According to the assumption, each map operation will take an equal amount of time, which is denoted by α. Each map task will be checked once it is finished. In this figure, T-1 represents the time slot to examine Map-1 through the A-test. The average A-test time is denoted by $\bar{\alpha}$. If $\bar{\alpha} < \alpha$, T_m is mainly determined by the Map process, otherwise T_m is mainly determined by the A-test process. Similar result can be obtained for the Reduce operation. We can formulate the OWAA problem as follows:

Find three-tuple <*a*, *m*, *r*> to

Minimize $T = T_m + T_s + T_r$ (3.1)

in which

0 < m < n, 0 < r < n, 0 < a, and *m*, *r*, *a* are integers.

$$T_m = \begin{cases} \left\lceil \frac{w_1}{a \cdot b} \right\rceil \cdot \bar{\alpha} + \alpha & \bar{\alpha} > \alpha \\ \left\lceil \frac{w_1}{m \cdot b} \right\rceil \cdot \alpha + \begin{cases} \bar{\alpha} & a \geq m, \bar{\alpha} \leq \alpha \\ \left(\left\lceil \frac{w_1}{a \cdot b} \right\rceil - \left\lceil \frac{w_1}{m \cdot b} \right\rceil + 1 \right) \cdot \bar{\alpha} & a < m, \bar{\alpha} \leq \alpha \end{cases} \end{cases} \quad (3.2)$$

$$T_s = f_s(m, w_2) \quad (3.3)$$

$$T_r = \begin{cases} \left\lceil \frac{w_2}{a \cdot b} \right\rceil \cdot \bar{\beta} + \beta & \bar{\beta} > \beta \\ \left\lceil \frac{w_2}{r \cdot b} \right\rceil \cdot \beta + \begin{cases} \bar{\beta} & a \geq r, \bar{\beta} \leq \beta \\ \left(\left\lceil \frac{w_2}{a \cdot b} \right\rceil - \left\lceil \frac{w_2}{r \cdot b} \right\rceil + 1 \right) \cdot \bar{\beta} & a < r, \bar{\beta} \leq \beta \end{cases} \end{cases} \quad (3.4)$$

$$h = \left\lceil \log_{(1-p_m)}^{(1-p_A)} \right\rceil \quad (3.5)$$

$$w_2 = f_w(w_1) \quad (3.6)$$

$$\alpha = f_\alpha(b) \quad (3.7)$$

$$\beta = f_\beta(b) \quad (3.8)$$

$$\bar{\alpha} = f_{\bar{\alpha}}(h) \quad (3.9)$$

$$\bar{\beta} = f_{\bar{\beta}}(h) \quad (3.10)$$

Equation (3.1) gives the objective function T (i.e., total processing time of a job), which consists of the Map phase time (i.e., T_m), the shuffle phase time (i.e., T_s), and the Reduce phase time (i.e., T_r). Equation (3.2) calculate T_m. Based on our analysis on Fig. 3.7, if $\bar{\alpha} > \alpha$, T_m is mainly determined by the A-test time, which is obtained from $\lceil w_1/(a \cdot b) \rceil \cdot \bar{\alpha}$. If $\bar{\alpha} < \alpha$, T_m is mainly determined by the Map time, which is calculated from $\lceil w_1/(m \cdot b) \rceil \cdot \alpha$. In addition, the number of auditors affects the calculation of T_m. If the number of auditors are no less than then

number of mappers (i.e., $a \geq m$), one $\bar{\alpha}$ is added into T_m (e.g., T-4 in Fig. 3.7); if $a < m$, each auditor will be assigned more A-test tasks, and the number of these extra A-test tasks can be calculated from $\lceil w_1/(a \cdot b) \rceil - \lceil w_1/(m \cdot b) \rceil + 1$. Similarly, we can obtain T_r. Equations (3.3), (3.6) - (3.10) are functions without concrete forms. To simplify the problem, we Assume the following functions are linear:

$$w_2 = f_w(w_1) = k_w \cdot w_1 \tag{3.11}$$

$$T_s = f_s(m, w_2) = k_s \cdot \frac{w_2}{m} \tag{3.12}$$

$$\alpha = f_\alpha(b) = k_\alpha \cdot b \tag{3.13}$$

$$\beta = f_\beta(b) = k_\beta \cdot b \tag{3.14}$$

$$\bar{\alpha} = f_{\bar{\alpha}}(h) = k_{\bar{\alpha}} \cdot h \tag{3.15}$$

$$\bar{\beta} = f_{\bar{\beta}}(h) = k_{\bar{\beta}} \cdot h \tag{3.16}$$

The coefficients of the above linear functions will be specified in evaluation.

3.7.2 Solve the OWAA problem

Accountability can be regarded as one type of quality of service that can be selected by customers in terms of P-Accountability. Therefore, when P-Accountability increases, it needs longer time to accomplish the job. Based upon the plans we designed, there are two scenarios in which the relations among *m*, *r*, and *a* differ.

Scenario 1: The auditors are dedicated testing machines that are not included in the customer group working machines (i.e., $m + r = n$.) In this case, the auditors are external to the customer working group. Therefore, with more auditors, the A-test will perform faster. To limit the number of auditors, we introduce a threshold a_0, i.e., $a \leq a_0$.

Scenario 2: The auditors are included in the customer group working machines (i.e., $m + r + a = n$). We consider the AG size as a key factor that affects the processing time. There are two aspects of the impact of AG size on the processing time: First, the AG occupies some computing resources that are supposed to run MapReduce tasks. In this regard, the AG slows down the system. On the other hand, the AC size determines the time of the A-test, which could be a major part of the total processing time. With a larger AC, the test will run faster. Therefore, there is a tradeoff of AC size.

Based on the formulation, we have four cases to obtain T:

*1)**Case A**: if* $\bar{\alpha} > \alpha$, $\bar{\beta} > \beta$

Combining equations (3.1) - (3.4), we have

$$T = \left\lceil \frac{w_1}{a \cdot b} \right\rceil \cdot \bar{\alpha} + \alpha + k_s \cdot \frac{w_2}{m} + \left\lceil \frac{w_2}{a \cdot b} \right\rceil \cdot \bar{\beta} + \beta \tag{3.17}$$

There are two sub-cases (i.e., A1 and A2), each representing a scenario:

A 1: If $m + r = n$, then all terms but $k_s \cdot \frac{w_2}{m}$ are relevant to m or r; therefore, when $m = n - 1$, $r = 1$, and $a = a_0$, we have a minimum of T as follows:

$$T_{\min} = \left\lceil \frac{w_1}{a_0 \cdot b} \right\rceil \cdot \bar{\alpha} + \alpha + k_s \cdot \frac{w_2}{n-1} + \left\lceil \frac{w_2}{a \cdot b} \right\rceil \cdot \bar{\beta} + \beta \tag{3.18}$$

A 2: If $m + r + a = n$, we have $r = 1$, and let $a = n - m - 1$. Then, equation (3.17) can be written as:

$$T = \left\lceil \frac{w_1}{(n-m-1) \cdot b} \right\rceil \cdot \bar{\alpha} + \alpha + k_s \cdot \frac{w_2}{m} + \left\lceil \frac{w_2}{(n-m-1) \cdot b} \right\rceil \cdot \bar{\beta} + \beta \tag{3.19}$$

Since $1 \le m \le n-2$, we can write (3.19) to the following:

$$T = \frac{c_1}{c_2 - m} + \frac{c_3}{m} + c_4 \tag{3.20}$$

in which $c_1 = \left(w_1 \cdot \bar{\alpha} + w_2 \cdot \bar{\beta}\right) \Big/ b$, $c_2 = n - 1$, $c_3 = k_s \cdot w_2$, $c_4 = \alpha + \beta$. Therefore, T is transformed to a function of single variable. Based on calculus, we have $T' = \frac{c_1}{(c_2 - m)^2} - \frac{c_3}{m^2}$. Let $T' = 0$, since we have $c_1 > 0$, $c_2 - m > 0$, $c_3 > 0$, and $m > 0$. By solving $T' = 0$, we have $m = m_0 = \frac{c_2 \cdot \sqrt{c_3}}{\sqrt{c_1} + \sqrt{c_3}}$. In addition, $T'' = \frac{2c_1}{(c_2 - m)^3} + \frac{2c_3}{m^3}$, $T''|_{m_0} > 0$, therefore, T can achieve minimum at this point. Since *m*, *r*, and *a* are integers. Therefore, we have:

- If $0 < m_0 < 1$, then <*m* = 1, *r* = 1, *a* = *n* - 2> is the optimal solution.
- If, $m_0 > n - 2$ then, <*m* = *n* - 2, *r* = 1, *a* = 1> is the optimal solution.
- If $1 \le m_0 \le n - 2$, then $< m = \begin{cases} \lceil m_0 \rceil & T(\lceil m_0 \rceil) < T(\lfloor m_0 \rfloor) \\ \lfloor m_0 \rfloor & T(\lceil m_0 \rceil) \ge T(\lfloor m_0 \rfloor) \end{cases}$, *r* = 1, *a* = *n* - *m* - 1> is the optimal solution.

*2)**Case B**: if $\bar{\alpha} > \alpha$, $\bar{\beta} < \beta$.*

Based upon the A-test scheme, this case is impossible because once p_A is determined, it has the same effect on A-test time for both Map and Reduce. Therefore, it can either be $\bar{\alpha} > \alpha$, $\bar{\beta} > \beta$, or $\bar{\alpha} < \alpha$, $\bar{\beta} < \beta$. We can remove both case B and case C for this reason.

*3)**Case C**: if $\bar{\alpha} < \alpha$, $\bar{\beta} > \beta$*

Based on the analysis on case B, case C is impossible to happen as well.

*4)**Case D**: if $\bar{\alpha} < \alpha$, $\bar{\beta} < \beta$*

There are four sub-cases, and each sub-case is discussed in two scenarios.

D1: If $m \le a$, $r \le a$, then

$$T = \left\lceil \frac{w_1}{m \cdot b} \right\rceil \cdot \alpha + \bar{\alpha} + k_s \cdot \frac{w_2}{m} + \left\lceil \frac{w_2}{r \cdot b} \right\rceil \cdot \beta + \bar{\beta} \tag{3.21}$$

D11: If $m + r = n$, T is not related to *a*, meaning that *a* can be as small as possible. We have $a = \min\{a_0, \max\{m, r\}\}$. T can be simplified as $T = \frac{p_1}{p_2 - m} + \frac{p_3}{m} + p_4$, where $p_1 = (w_2 \cdot \beta)/b$, $p_2 = n$, $p_3 = (w_1 \cdot \alpha)/b + k_s \cdot w_2$, and $p_4 = \bar{\alpha} + \bar{\beta}$. Similar to case *A12*, when we have $m = m_0 = \frac{p_2 \cdot \sqrt{p_3}}{\sqrt{p_1} + \sqrt{p_3}}$, T can achieve minimum. Then,

- If $0 < m_0 < 1$, then $< m = 1$, $r = n - 1$, $a = \min\{a_0, n-1\} >$ is the optimal solution.
- If $m_0 > n - 2$, then, $<m = n$-1, $r = 1$, $a = \min\{a_0, n-1\} >$ is the optimal solution.
- If $1 \le m_0 \le n - 2$, then $< m = \begin{cases} \lceil m_0 \rceil & T(\lceil m_0 \rceil) < T(\lfloor m_0 \rfloor) \\ \lfloor m_0 \rfloor & T(\lceil m_0 \rceil) \ge T(\lfloor m_0 \rfloor) \end{cases}$, $r = n - m$, $a = min\ \{a_0, max\{m, r\}\}>$ is the optimal solution.

D12: If $m + r + a = n$, T can be simplified as

$$T = \frac{p_1}{p_2 - a - m} + \frac{p_3}{m} + p_4 \tag{3.22}$$

Therefore, T is a function of two variables. To find T_{min}, we can compute the following: $\frac{\partial T}{\partial a} = \frac{p_1}{(p_2 - m - a)^2}$, $\frac{\partial T}{\partial m} = \frac{p_1}{(p_2 - m - a)^2} - \frac{p_3}{m^2}$, $\frac{\partial^2 T}{\partial a^2} = \frac{2p_1}{(p_2 - m - a)^3}$, $\frac{\partial^2 T}{\partial m^2} = \frac{2p_1}{(p_2 - m - a)^3} + \frac{2p_3}{m^3}$, and $\frac{\partial T}{\partial a \partial m} = -\frac{2p_1}{(p_2 - m - a)^3}$. Let $\frac{\partial T}{\partial a} = 0$, and $\frac{\partial T}{\partial m} = 0$, we have $\begin{cases} \frac{p_1}{(p_2 - m - a)^2} = 0 \\ \frac{p_1}{(p_2 - m - a)^2} - \frac{p_3}{m^2} = 0 \end{cases}$. Since $p_1 > 0$, there is no solution of the above equation set. Therefore, there is no extreme point for *T*.

We conclude that *T* is at its minimum when the following statements hold: 1) $m/r = \sqrt{p_3}/\sqrt{p_1}$, 2) $m + r + a = n$, 3) $m \le a$, $r \le a$, 4) a can be as small as possible. The optimal solution is $m = \left\lfloor \min\left(\frac{n}{2+\sqrt{p_1/p_3}}, \frac{n}{1+2\sqrt{p_1/p_3}}\right) \right\rfloor$, $r = \left\lfloor m\sqrt{p_1/p_3} \right\rfloor$, and $a = n - m - r$.

D2: if $r > a \ge m$, then $T = \left\lceil \frac{w_1}{m \cdot b} \right\rceil \cdot \alpha + \bar{\alpha} + k_s \cdot \frac{w_2}{m} + \left\lceil \frac{w_2}{r \cdot b} \right\rceil \cdot \beta + \left(\left\lceil \frac{w_2}{a \cdot b} \right\rceil - \left\lceil \frac{w_2}{r \cdot b} \right\rceil + 1 \right) \cdot \bar{\beta}$

D21: If $m + r = n$, *T* can be transformed to a function $T(a,m) = \frac{q_1}{n-m} + \frac{q_2}{m} + \frac{q_3}{a} + q_4$, where $q_1 = \frac{w_2}{b}(\beta - \bar{\beta})$, $q_2 = \frac{w_1 \cdot \alpha}{b} + k_s \cdot w_2$, $q_3 = \frac{w_2 \cdot \bar{\beta}}{b}$, $q_4 = \bar{\alpha} + \bar{\beta}$, $\frac{\partial T}{\partial a} = -\frac{q_3}{a^2}$, $\frac{\partial T}{\partial m} = \frac{q_1}{(n-m)^2} - \frac{q_2}{m^2}$, $\frac{\partial^2 T}{\partial a^2} = \frac{q_3}{a^3}$, $\frac{\partial^2 T}{\partial m^2} = \frac{q_1}{(n-m)^3} + \frac{q_2}{m^3}$, and $\frac{\partial T}{\partial a \partial m} = 0$. Let $\frac{\partial T}{\partial a} = 0$, and $\frac{\partial T}{\partial m} = 0$, we cannot find a solution for *a* and *m*, meaning that there is no extreme point of T. We then conclude that *T* is at its minimum when the following statements hold: 1) *a* is as large as possible but $a \le a_0$, 2) $r > a \ge m$, 3) $\frac{\partial T}{\partial m} = 0$, from which we have $m = m_0 = \left(n\sqrt{q_2}\right)/\left(\sqrt{q_1} + \sqrt{q_2}\right)$. We can then determine the optimal solution of T:

- If $0 < m_0 < 1$, then $< m = 1,\ r = n-1,\ a = \min\{a_0, r-1\} >$ is optimal.
- If $m_0 > n-2$, then, <m = n-2, r = 2, $a = \min\{a_0, 1\}$ > is optimal.
- If $1 \le m_0 \le n-2$, then $< m = \begin{cases} \lceil m_0 \rceil & T(\lceil m_0 \rceil) < T(\lfloor m_0 \rfloor) \\ \lfloor m_0 \rfloor & T(\lceil m_0 \rceil) \ge T(\lfloor m_0 \rfloor) \end{cases}$, $r = n - m$,

  $a = \min\{a_0, r-1\}$ > is optimal.

D22: If $m + r + a = n$, we have $T(a,m) = \frac{q_1}{n-m-a} + \frac{q_2}{m} + \frac{q_3}{a} + q_4$, then $\frac{\partial T}{\partial a} = \frac{q_1}{(n-m-a)^2} - \frac{q_3}{a^2}$, $\frac{\partial T}{\partial m} = \frac{q_1}{(n-m-a)^2} - \frac{q_2}{m^2}$, $\frac{\partial^2 T}{\partial a^2} = \frac{2q_1}{(n-m-a)^3} - \frac{2q_3}{a^3}$, $\frac{\partial^2 T}{\partial m^2} = \frac{2q_1}{(n-m-a)^3} - \frac{2q_2}{m^3}$, and $\frac{\partial T}{\partial a \partial m} =$

$\frac{2q_1}{(n-m-a)^3}$. Let $\frac{\partial T}{\partial a}=0$, and $\frac{\partial T}{\partial m}=0$, we have $\begin{cases}\frac{q_1}{(n-m-a)^2}-\frac{q_3}{a^2}=0\\ \frac{q_1}{(n-m-a)^2}-\frac{q_2}{m^2}=0\end{cases}$, by solving the equation set, we have

$$\begin{cases}m=m_0=n\cdot\sqrt{\frac{q_2}{q_1+q_2+q_3}}\\ a=a_0=n\cdot\sqrt{\frac{q_3}{q_1+q_2+q_3}}\end{cases}.$$

Let $F(a,m)=\left(\frac{\partial T}{\partial a\partial m}\right)^2-\left(\frac{\partial^2 T}{\partial a^2}\right)\cdot\left(\frac{\partial^2 T}{\partial m^2}\right)$, then we can compute $F(a_0,m_0)<0$, and $\frac{\partial^2 T}{\partial a^2}|_{a_0}>0$. Therefore, T can achieve its minimum. The optimal solution for this case is: If $a_0 \geq m_0$, and $r=n-m_0-a_0>a_0$, then m will be $\lfloor m_0 \rfloor$ or $\lceil m_0 \rceil$, a will be $\lfloor a_0 \rfloor$ or $\lceil a_0 \rceil$, and $r = n - m - a$. Otherwise, there is no optimal solution.

D3: if $m>a\geq r$, then $T=\left\lceil\frac{w_1}{m\cdot b}\right\rceil\cdot\alpha+\left(\left\lceil\frac{w_1}{a\cdot b}\right\rceil-\left\lceil\frac{w_1}{m\cdot b}\right\rceil+1\right)\cdot\bar{\alpha}+k_s\cdot\frac{w_2}{m}+\left\lceil\frac{w_2}{r\cdot b}\right\rceil\cdot\beta+\bar{\beta}$

D31: If $m+r=n$, T can be simplified as $T(a,m)=\frac{s_1}{n-m}+\frac{s_2}{m}+\frac{s_3}{a}+s_4$, which is similar to case D21. Let $m=m_0=\left(n\sqrt{s_2}\right)/\left(\sqrt{s_1}+\sqrt{s_2}\right)$, we can determine the optimal solution as follows:

- If $m_0<n-m_0$, there is no optimal solution.
- If $m_0\geq n-m_0$, then $<m=\begin{cases}\lceil m_0\rceil & T(\lceil m_0\rceil)<T(\lfloor m_0\rfloor\\ \lfloor m_0\rfloor & T(\lceil m_0\rceil)\geq T(\lfloor m_0\rfloor\end{cases}$, $r=n-m$, a = min {a_0, m-1}> is the optimal solution.

D32: If $m+r+a=n$, then T can be simplified as $T(a,m)=\frac{s_1}{n-m-a}+\frac{s_2}{m}+\frac{s_3}{a}+s_4$, where $s1=\frac{w_2\cdot\beta}{b}$, $s_2=\frac{w_1}{b}\cdot(\alpha-\bar{\alpha})+k_s\cdot w_2$, $s_3=\frac{w_1\cdot\bar{\alpha}}{b}$, $s_4=\bar{\alpha}+\bar{\beta}$. This case is similar to D22.

D4: if $m > a$, $r > a$, then $T = \left\lceil \frac{w_1}{m \cdot b} \right\rceil \cdot \alpha + \left(\left\lceil \frac{w_1}{a \cdot b} \right\rceil - \left\lceil \frac{w_1}{m \cdot b} \right\rceil + 1 \right) \cdot \bar{\alpha} + k_s \cdot \frac{w_2}{m} + \left\lceil \frac{w_2}{r \cdot b} \right\rceil \cdot \beta +$

$\left(\left\lceil \frac{w_2}{a \cdot b} \right\rceil - \left\lceil \frac{w_2}{r \cdot b} \right\rceil + 1 \right) \cdot \bar{\beta}$. T can be simplified as $T(a,m) = \frac{g_1}{r} + \frac{g_2}{m} + \frac{g_3}{a} + g_4$, where $g_1 = \frac{w_2}{b} \cdot (\beta - \bar{\beta})$,

$g_2 = \frac{w_1}{b} \cdot (\alpha - \bar{\alpha}) + k_s \cdot w_2$, $g_3 = \frac{w_1 \cdot \bar{\alpha}}{a} + \frac{w_2 \cdot \bar{\beta}}{b}$, $g_4 = \bar{\alpha} + \bar{\beta}$.

D41: If $m + r = n$, the case is similar to case D21. Let $m = m_0 = \left(n\sqrt{g_2} \right) \Big/ \left(\sqrt{g_1} + \sqrt{g_2} \right)$, we can determine the optimal solution as follows:

- If $0 < m_0 < 1$, there is no optimal solution.
- If $m_0 > n - 2$, there is no optimal solution.
- If $1 \le m_0 \le n - 2$, then $< m = \begin{cases} \lceil m_0 \rceil & T(\lceil m_0 \rceil) < T(\lfloor m_0 \rfloor \\ \lfloor m_0 \rfloor & T(\lceil m_0 \rceil) \ge T(\lfloor m_0 \rfloor \end{cases}$, $r = n - m$, a = min {a_0, min(m, r)} > is the optimal solution.

D42: If $m + r + a = n$ this case is similar to case D22.

The problem set and its solutions can be summarized in Table 3.4.

Table 3.4 Solution table

Condition / Case #		$m + r = n$	$m + r + a = n$
A: $\bar{\alpha} > \alpha$, $\bar{\beta} > \beta$		√	√
B: $\bar{\alpha} > \alpha$, $\bar{\beta} < \beta$		N/A	N/A
C: $\bar{\alpha} < \alpha$, $\bar{\beta} > \beta$		N/A	N/A
D: $\bar{\alpha} < \alpha$, $\bar{\beta} < \beta$	$m \le a$, $r \le a$	√	√
	$r > a \ge m$	√	√
	$m > a \ge r$	√	√
	$m > a$, $r > a$	√	√

Taking WordCount as the host application, the parameters can be specified as follows:

Table 3.5 Default parameter settings

T_0	20 hr	b	64 MB
w_1	2 TB	p_m	0.1
p_A	0.9	n	100
k_w	1.4	k_s	10^{-4}
k_α	0.16	k_β	0.23
$k_{\bar{\alpha}}$	10^{-4}	$k_{\bar{\beta}}$	10^{-4}

For WordCount, each 'word' will be transformed to a key-value pair like <'word', 1>, meaning that 2 letters (i.e., ',' and '1') are added. We assume that the average length of an English word is 5; the size of each word will be increased by 2/5, and we have $k_w = 7/5 = 1.4$.

Table 3.6 Numeric results

Parameter \ Condition		$m+r=n$			$m+r+a=n$		
		Case	<*m*,*r*,*a*>	Tmin	Case	<*m*,*r*,*a*>	Tmin
default		D11	<41,59,59>	5.2	D42	<57,39,4>	6.25
P_A	0.5	D11	<41,59,59>	5.2	D42	<57,41,2>	5.99
	0.999	D11	<41,59,59>	5.2	D42	<57,36,7>	6.7
	1	A1	<99,1,50>	5.89	A2	<1,1,98>	8.32
N	50	D11	<20,30,30>	10.4	D42	<28,20,2>	12.34
B	128	D11	<41,59,59>	5.2	D42	<57,41,2>	6.04
w_1	8	D11	<41,59,59>	20.2	D42	<57,39,4>	25.03

The following figures describe how T changes in different cases when parameters are using default values. Fig. 3.8 shows case A2 with default setting. In this case, we have $T = \frac{1060400}{99-m} + \frac{280}{m} + 24.96$. when m increases, T keeps increasing. Therefore, <$m = r = 1$, $a = n - m - r$> is the optimal solution. Fig. 3.9 shows case D11 with default setting. In this case, we have $T = \frac{644000}{100-m} + \frac{320280}{m} + 0.046$, T can achieve minimum, which is 5.2 hr. Fig. 3.10 shows

case D42 with default setting, we have $T(a,m) = \frac{642993}{100-m-a} + \frac{319561}{m} + \frac{1725}{a} + 0.046$, and T_{min} = 6.25 hr.

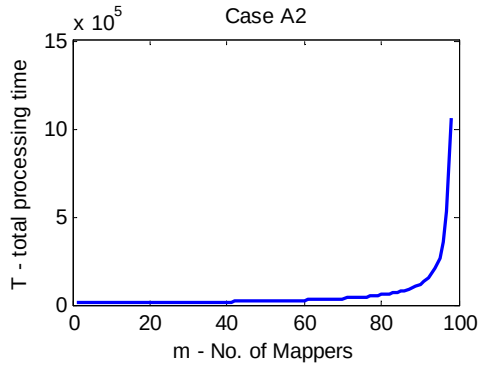


Fig. 3.8 Case A2 with default setting

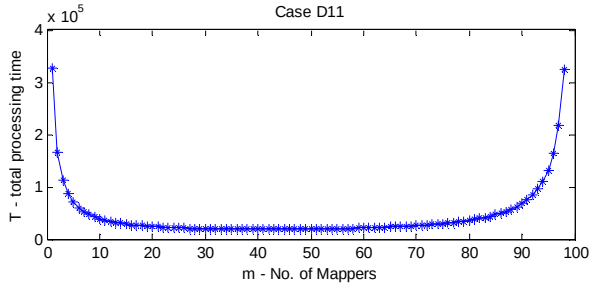


Fig. 3.9 Case D11 with default setting

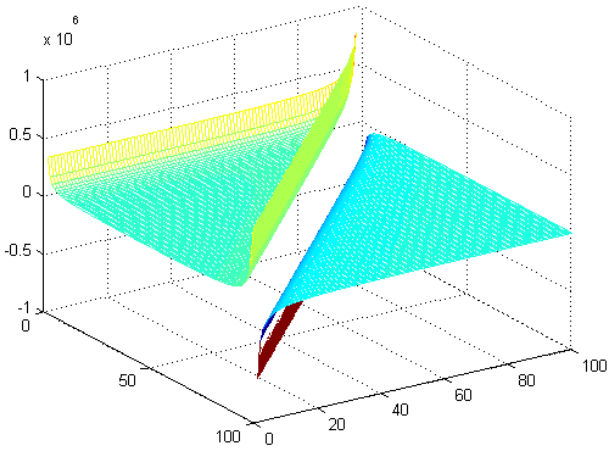


Fig. 3.10 Case D42 with default setting

3.8 Evaluation

3.8.1 Experiment Setup

We implement a prototype of Accountable MapReduce based upon Hadoop [33] and have tested it in both our local lab and the Utah Emulab testbed [76]. We set up a Virtual LAN (VLAN) with 20 PCs in Emulab to deploy the prototype of Accountable MapReduce. The MapReduce application is Wordcount.

3.8.2 Experiment Result

Fig. 3.11 depicts how the AG size affects the processing time when P_A = 1. Because the purpose is to test how the A-test will affect processing time, we do not insert any malicious nodes. We build a 5-node cluster to run MapReduce, and we change the size of AG to observe how the processing time would change accordingly. Our result shows that when there are no auditors (AG#=0), the MapReduce system is not accountable, but the processing time is minimal because there is no extra overhead added by the A-test. The increase of the AG size will bring extra overhead to the job. Since P_A=1 in this case, the job will be duplicated. The more auditors we have, the quicker the A-test will finish. A straightforward observation is that when the AG size is equal to the number of workers, each worker will be tested by an individual auditor so that some waiting time will be saved.

Fig. 3.12 shows how P-Accountability affects the processing time. We also reduce P_A from 1 to 0.99 to observe how this change will affect the processing time. We use a 5-node cluster to run Wordcount (data size = 50M). In order to rule out the interference of re-submit/re-process time, this experiment is also free of malicious nodes. The result shows that when P_A is reduced, the workload of A-test is significantly reduced. As a result, each mapper/reducer will be

tested very quickly with a slightly smaller P_A. Also, the AG size will not become an issue since equivalent performance can be achieved with fewer auditors.

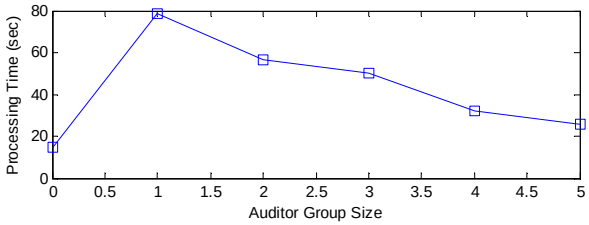


Fig. 3.11 AG Size and Processing Time When P_A = 1

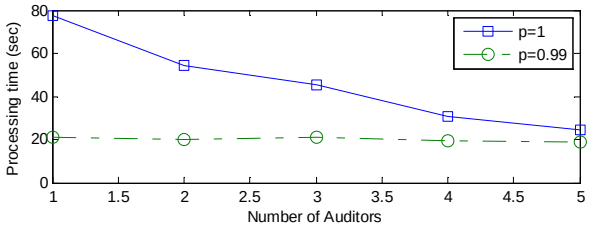


Fig. 3.12 P-Accountability and Processing Time

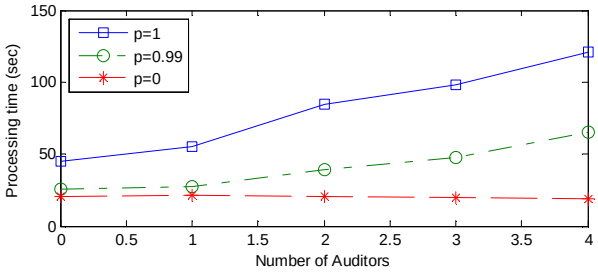


Fig. 3.13 Malicious nodes and processing time

When malicious nodes are taken into account, we need to add the re-submit/re-process/re-test time into the total processing time. In this experiment, the cluster still contains 5 workers. The AG size is 2, and the input data is 100MB. Fig. 3.13 shows the performance of Accountable MapReduce when P_A=0, 0.99 and when P_A = 1. In the chart, the gap between the two curves means that the test time is greatly reduced. Also, we observe that the more malicious nodes there are, the longer the processing time will be, because once malicious nodes are

detected, they will be exposed and no longer take tasks. As a result, the remaining good workers will run the tasks that need to be reprocessed, and the AG will re-test the tasks.

False positives may exist when P_A is less than 1 because the map/reduce task will not be fully tested. However, based upon our assumption that a malicious node randomly manipulates results, then they can be detected with high probability. Another point is that the performance is greatly improved when P_A is close to 1 (e.g., P_A=0.999). In our experiment (20 repetitions), we found that when P_A = 0.99 and p_m = 0.01, the A-test catches every malicious node. We also tested an extreme case in which P_A = 0.99 and p_m = 0.0001, meaning that the malicious node may alter one record out of every 10000 records. In this case, we did observe false positives.

3.9 Conclusion

In this chapter, we propose Accountable MapReduce as an additional component for the current MapReduce model to support accountability. The Accountable MapReduce employs an Auditor Group to conduct the A-test on every worker in real time. If malicious behavior occurs, the AG is able to detect it and provide verifiable evidence. To improve the performance, we introduce P-Accountability in the A-test to trade the degree of accountability with efficiency. We implement a prototype of Accountable MapReduce in the Hadoop platform. Our evaluation results show that our scheme can be practically and efficiently utilized in cloud systems.

CHAPTER 4

NON-REPUDIATION IN NEIGHBORHOOD AREA NETWORKS

FOR SMART GRID

4.1 Introduction

The Smart Grid [48 - 50] has recently become one of the research hotspots. The Smart Grid not only delivers electricity from the power provider to users, it also enables two-way digital communications to gather, distribute, and act on information about the behavior of all participants. The goal of replacing traditional power grids with smart grid is to save energy, to reduce cost, and to increase reliability and transparency.

The traditional power grid does not possess the property of non-repudiation for meter reading. Back in the 20th century, utility companies employed meter readers to do a door-to-door meter reading. The drawbacks of artificial meter reading include high time cost and labor cost, low accuracy, error-prone reading, etc. Additionally, there is no evidence pointing to the cheater who falsifies or manipulates the reading data. For instance, if a meter is compromised and the reading is less than the actual amount of service, the power company is unable to detect the theft behavior. Advanced Metering Infrastructure (AMI) is developed to tackle some of these issues. The goal of AMI is to provide automatic measurement and transmission of the meter reading by integrating smart meters with digital communication techniques. However, AMI could not ensure non-repudiation of meter reading as well. The root problem lies in the way of collecting the reading value. In order to acquire the amount of service for each user, the power provider has

to rely on the digital communication network for data transmission. Since the reading value is generated on the user end, an attacker or an energy thief still have multiple means to tamper with it. The most common methods [51] of energy theft include: metering tampering, meter switching, wire partial bypass of the meter inside the meter enclosure, complete bypass of the meter from the low-voltage grid, and direct connection to the primary voltage grid with a pirate distribution transformer. Most of these methods are not applicable to the Smart Grid. However, new methods have been developed to compromise a smart meter. For example, researchers have successfully hacked into a smart meter with a recovered backdoor account [30], and then tampered with the reading. Since the smart meter may be the only source of the amount of service, the power provider has no means to verify the correctness of the meter's reading report.

A straightforward solution is to physically secure the smart meter in a safe box. People who attempt to break the box may trigger an alarm or leave an undeniable trace on the box. However, this does not solve the root problem: the amount of service can only be obtained via the meter on the user end. In addition, recent research shows a possibility of the manipulation of reading value in the air [53], which makes this solution less promising.

In this chapter, we address the non-repudiation issue with accountability which assigns responsibility to each smart meter. We adopt a mutual-inspection strategy to ensure non-repudiation. Following this strategy, we install two smart meters for one power line connecting the user and the provider. This means that for each individual power line, there is one smart meter on each end; one represents the user's reading, and the other represents the provider's reading. In a normal situation, although these two meters measure the same power line, their readings are not exactly the same due to 1) power loss during transmission, 2) measuring errors caused by communication delays and synchronization, and 3) dynamic factors caused by the

environment (e.g., temperature). Additionally, the remarkable difference between readings may infer a malicious/malfunctioning meter. In our strategy, the readings are exchanged between power provider and the user in order to resolve dispute (if there is any). If the dispute lies in a range that is acceptable for both ends, the service continues to be delivered; otherwise, the service will be terminated and further investigation will launch. Therefore, two distrusted parties can inspect each other to achieve non-repudiation in the Smart Grid.

Mutual inspection requires doubling the quantity of smart meters, leading to a high expense. However, the estimated annual loss due to energy theft is 6 billion dollars in the United States [51]. There are around 22 million smart meters deployed in the US by the end of 2011, and the market price of a smart meter is around $100. To fully adopt mutual inspection, the number of smart meters will be doubled. Therefore, the hardware cost will be less than 3 billion. In addition, the cost of deployment and maintenance should be considered. If the strategy saves the 6-billion-dollar loss or at least the majority, the saved money in one year may entirely cover the investment. The Return on Investment is considerable because the country can save up to 6 billion dollars per year in the future years.

The contributions of this chapter are listed as follows:

- We address the non-repudiation problem for the Smart Grid. Based on our knowledge, this is the first time the non-repudiation problem is discussed in the Smart Grid.
- We adopt a mutual-inspection and design a protocol to realize non-repudiation so that any misbehavior that tampers with power readings will be detected. Mutual-inspection is able to detect all theft behaviors relying on meter compromising and wire bypassing.

- We consider three kinds of structures: a centralized structure, a Peer-to-Peer (P2P) structure, and a hybrid structure, and we discuss how mutual inspection is applied to the three environments.
- We conduct both numeric analysis and simulation to evaluate the proposed scheme.

4.2 Related Work

Security has been a significant concern for the Smart Grid [52 - 54, 56 - 58, 62, 67 - 75]. The Smart Grid has leveraged many hardware and software technologies, such as smart meters, sensors, and advanced communication networks. Although these techniques bring many exciting features to the Smart Grid, they also introduce new vulnerabilities that may be exploited by adversaries. We briefly introduce the smart grid security issues in four aspects:

- **Trust** – the Smart Grid is a heterogeneous environment containing various devices, such as smart meters, appliances, collectors, and backend servers. A trust relation is expected before real data is exchanged and processed. The issue discussed in this chapter falls into this category.
- **Privacy** – smart metering and load management is incorporated into the Smart Grid. However, since the user's load profile is revealed to the provider, the user's privacy, especially the living style, may be disclosed. For instance, it is easy to tell whether the customer is at home or not. With careful analysis, it is even possible to figure out which appliance is in use. This kind of information may be used by criminals or third parties with malicious purposes.

- **Device security** – devices in the Smart Grid should be protected physically and cryptographically. Otherwise, attackers may compromise a smart meter or other devices to obtain cipher keys and memory data.
- **Security management** – the scale of the Smart Grid keeps increasing; thus more and more devices will join in. Handling the security management issues such as key generation, update, and revocation in a large-scale environment is another challenge.

In this chapter, our focus is on non-repudiation, which is the foundation of the trustworthy Smart Grid. McLaughlin *et al*. [53] demonstrate that not only energy theft is possible in the Advanced Metering Infrastructure (AMI) system, but that the AMI commodity devices can be exploited by adversaries in order to perform a number of attacks. There are three classes of attacks depending on when and where meter reading is manipulated. They include: 1) while it is recorded, 2) while it is at rest in the meter, and 3) as it is in flight across the network. Today's energy theft detection models generally fall into two categories [54]: peer comparison and characteristic analysis. Peer comparison models group residential and commercial customers with similar homes and businesses in similar geographical and environmental settings. If a customer's actual usage deviates from the expected usage, it may indicate anomaly on the reading. Characteristic analysis, on the other hand, attempts to model the consumption pattern for an account; thus, any anomalies not following the pattern may indicate energy theft. In this area, machine learning techniques (e.g., SVMs [55]) can be leveraged for building fundamental patterns and detecting anomalies. However, these analytical methods cannot be used as evidence of energy theft, because a deviation from expected normal usage can be caused by multiple reasons other than energy theft. For example, one needs to consider the trend of energy usage in

the entire area or other reasonable changes. In this chapter, our method differs from the early analytical methods in that we aim to isolate the compromised meter(s) with undeniable evidence, which can be used as proof of misbehavior.

Accountability has been a longstanding concern for trustworthy computer systems [2], and it has recently been elevated to a first class design principle for dependable networked systems [3]. In general, accountability implies an entity's capacity to identify a party that is responsible for specific events with undeniable evidence. Regarding the Smart Grid, relevant study on accountability is limited. Liu *et al*. [29, 56] have addressed accountability as a solution to build trustworthy Smart Grid in the Home Area Network (HAN), where 1) the smart meter and the smart appliance group are able to verify the correctness of each other, and 2) a power company can prove the correctness of the smart meter.

In this chapter, we focus on the Smart Grid in the Neighborhood Area Network (NAN) where the power company and independent users do not trust each other. We adopt a mutual-inspection to realize non-repudiation so that any misbehavior that alters the meter reading will be eventually detected.

4.3 Smart Grid in NAN

As shown in Fig. 4.1, there are two basic parties for the Smart Grid in NAN: the power company and independent users. An independent user may own a power generator; therefore, it can be both a power provider and a user. All entities are connected by two types of flows, i.e., electricity flow and information flow.

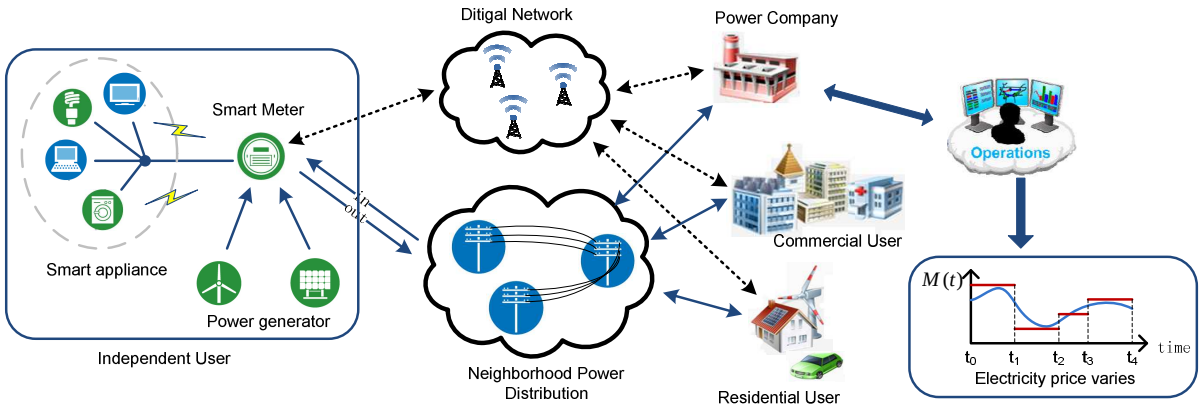


Fig. 4.1 Overview of Smart Grid in the Neighborhood Area Network

4.3.1 Pricing

Theoretically, price is changing in real time in the Smart Grid. The price is mainly determined by the supply and demand of power in a certain area. In real world scenarios, however, the price will usually be segmented. For example, the power company may divide a day into several time segments, each of which corresponds to a certain price. In the future, the length of a time segment may be significantly reduced to adopt a fine-grained pricing scheme.

4.3.2 Smart Grid Structure and Billing

The Cisco brief [58] has mentioned a few security challenges for the Smart Grid, and one of them is "the integration of distributed energy suppliers such as independent power producers, of renewable energy generation, and of distributed energy resources". This means that independent power producers are playing a more important role. Moreover, the billing mechanism in the future will change accordingly. Based on the way that how utility companies and independent users choose to participate in the Smart Grid, there are three structure options for the Smart Grid in NAN.

- *Centralized structure*: In the centralized structure, the utility company is the major power provider, and it also determines the market price $M(t)$. Every user follows

this price. Let $E_i(t)$ denote the amount of power consumed by user *i*, and let $S_i(t)$ be the power generated and sold by user *i*. The billing function $B_i(t_0, \Delta t)$ gives the total bill of user *i* during the time interval $[t_0, t_0 + \Delta t]$. In the centralized structure, $B_i(t_0, \Delta t) = \int_{t_0}^{t_0+\Delta t} M(t) \cdot \left(E_i(t) - S_i(t)\right) dt$. Since $M(t)$, $E_i(t)$, and $S_i(t)$ are functions of *t*, the bill can be calculated in terms of the integral on $M(t) \cdot \left(E_i(t) - S_i(t)\right)$.

- *P2P structure*: In the pure Peer-to-Peer (P2P) Smart Grid, every independent user acts as both a power provider and a user, and users can determine their own prices. Let $m_i(t)$ denote the price function given by user *i*, and let $E_{i,j}(t)$ denote the service amount user *i* bought from provider *j*. If there are *n* other users from which user i buys power during time window $[t_0, t_0 + \Delta t]$, then we have $B_i(t_0, \Delta t) = \int_{t_0}^{t_0+\Delta t} \left(\sum_{j=1}^{n} m_j(t) \cdot E_{i,j}(t) - m_i(t) \cdot S_i(t) \right) dt$. Therefore, the bill that user *i* pays to its provider *j* is given as $B_{i,j}(t_0, \Delta t) = \int_{t_0}^{t_0+\Delta t} \left(m_j(t) \cdot E_{i,j}(t) - m_i(t) \cdot S_{i,j}(t) \right) dt$.
- *Hybrid structure*: In the hybrid structure, the power company is still the major provider and determines the market price; however, independent users can set their own prices as well. Therefore, it is straightforward to give the billing function as $B_i(t_0, \Delta t) = \int_{t_0}^{t_0+\Delta t} (M(t) \cdot E_i(t) + \sum_{j=1}^{n} m_j(t) \cdot E_{i,j}(t) - m_i(t) \cdot S_i(t)) dt$. In this case, the total bill of user *i* should be the sum of the amount paid to the power company (the first term in parenthesis) and the amount paid to other independent

users (the second term), and minus the money that user *i* make from other users. If we treat the power company as another independent user, the hybrid structure becomes a special form of the P2P structure.

4.4 Problem statement

4.4.1 Terms and Definitions

Based on the mutual-inspection strategy, for a user *S*, there are two meters, K_S and K_P, to record its amount of service on both ends of the power line connecting the user *S* and the provider *P*.

Definition 4.1: *Power demand function* $P(t)$ gives the demand of a user at time *t*. Obviously, $P(t)$ depends on the condition of all power appliances. A concrete model of $P(t)$ will be given in the next section.

Definition 4.2: *Power loss function* $l(P)$ defines the power line loss during transmission given that the power demand is *P*. According to electricity, we have $l(P(t)) = \left(P(t)^2 \cdot R\right) \big/ V^2$, where *R* is the wire resistance and *V* is the transmission voltage. *R* is dependent on the cable material, cable length, and environment factors like temperature. In this chapter, we rule out the variability of resistance and consider *R* as a constant. Therefore, *l* is a function of power demand *P*.

Definition 4.3: *Bill difference (i.e., dispute) function* $b(t_0, \Delta t)$ gives the bill difference (starting from t_0) between two end smart meters connected by the same power line during a time window Δt. Let *M(t)* denote the price function, which can be either the market price or the independent power seller's price. Therefore, $b(t_0, \Delta t)$ can be given as $b(t_0, \Delta t) =$

$\frac{R}{V^2}\int_{t_0}^{t_0+\Delta t} M(t)\cdot P(t)^2 dt + \alpha(t_0,\Delta t)$. The first term is the bill calculated from the power loss, which is considered as the main part of the bill difference. However, there are other factors affecting the bill difference, including measuring error, communication delay, time synchronization issue, etc. Therefore, before we can have further information to express these factors, we use $\alpha(t_0,\Delta t)$ to represent all of them.

4.4.2 Problem Formulation

In order to resolve the dispute, K_S and K_P exchange the billing data constantly; however, there should be a proper time window that determines how frequent the billing data is exchanged. If the time window is too large (e.g., once per month/day/hour), then the power provider undertakes a higher risk because there may be a large difference accumulated before the provider can detect it. On the contrary, if the time window is short, the power provider can be notified before it suffers more loss. However, a short time window may incur high communication overhead. Therefore, it is essential to determine an optimal time window in order to maximize the time window, and to avoid huge bill difference. The Time Window Maximization (TWM) problem is formulized as follows.

Time Window Maximization (TWM) problem: given t_0, maximize Δt, s.t. $b(t_0,\Delta t)\le b_0$; $H(\Delta t) > H_0$, in which b_0 is a predefined dispute threshold. $H(\Delta t)$ is the throughput function that is constrained by a threshold H_0.

4.4.3 Solution Sketch

Problem TWM aims at maximizing Δt with two constraints. An intuitive solution to this problem is that the maximal Δt can be determined by the intersection of the solution sets for the two constraints. Suppose that the constraints $b(t_0,\Delta t)\le b_0$ and $H(\Delta t) > H_0$ have solution sets W_1

and W_2, respectively. If $W_1 \cap W_2 = W_* = \varnothing$, then no optimal Δt can be found. This means that b_0, or H_0 should be adjusted in order to generate a satisfactory Δt. If $W_1 \cap W_2 = W_* \neq \varnothing$, then an optimal Δt can be obtained. Considering the opposite influence of Δt on bill difference and throughput, we can be sure that an optimal Δt always exists if $W_1 \cap W_2 \neq \varnothing$.

4.5 Design of Mutual Inspection for the Smart grid in NAN

4.5.1 Protocol Overview

We design a protocol to ensure that if the actual bill difference between two smart meters K_P and K_S exceeds a threshold b_0, the trust relationship will break and the service will be terminated immediately.

There are only two roles in this protocol. They are smart meter K_P (representing the provider reading) and smart meter K_S (representing the user reading). The power provider and the user do not trust each other because the smart meters may be compromised.

4.5.2 Protocol Detail

We assume that one user can only have one power provider during a certain amount of time aside from the electricity produced by home power generators. We also assume that the AMI system has already employed Public-Key Infrastructure (PKI) to establish the authentication framework. Under this assumption, there is a Certificate Authority (CA) acting as a trusted third party.

When a new independent user *k* joins the Smart Grid, it follows the protocol until the electricity is cut off or until the service is shut down due to the detection of anomaly. The protocol can be described as follows:

1. User *k* joins in the Smart Grid by registering itself at the CA, and then starts to produce electricity with the power generator. User *k* needs to request a certificate from the CA for authentication purpose.
2. If the self-produced electricity cannot meet the power demand, user *k* becomes a power user that will find a power provider in the neighborhood and will then negotiate a bill difference threshold b_0 with its power provider (i.e., power company in the centralized architecture, other independent users who have extra electricity to sell in the P2P/hybrid architecture). Once the provider is determined, an authentication process will be initiated by the user via a handshake protocol. The outcome of the authentication process is that a) the two parties are authenticated to each other, and b) a secret key is securely distributed between the user and the provider for the future use of encryption/hashing. For example, the Security Socket Layer (SSL) protocol will suffice, since it establishes a secure channel between two parties to ensure multiple security properties.
3. If the self-produced electricity is more than the power demand, user *k* becomes a power provider because it has extra power to sell. The action to be taken depends on the type of structure:
    a. In the centralized structure, user *k* follows the price made by the utility company. It only needs to send extra electricity back to the power grid. Essentially, the power company buys the electricity from some home users and then sells it to other ones.

b. In the P2P/hybrid structure, user k broadcasts its own price to the entire neighborhood. If there is a request to subscribe, user k will start delivering power service after authentication.

4. Both the power provider and the user will maintain a service record, in which each entry is a four-tuple e_i = <type, ts_i, $R_k(ts_i)$, partner>. 'Type' indicates the role of the record owner (i.e., provider/user); ts_i is the timestamp of entry i; $R_k(ts_i)$ is the incoming/outgoing reading measured by the meter k at ts_i; 'partner' indicates the other side of the service. Based on the record, the provider and the user are able to compute the bill during a period. Given two entries e_i and e_j ($i < j$), in order to calculate the bill, there is one condition: fields 'type' and 'partner' do not change in entries from e_i up to e_j. This condition ensures that service is continuously delivered during ts_i and ts_j. We let $b(e_i, e_j)$ denote the bill.
5. After the bill is computed, a billing message will be constructed as follows: $M_{bill} = b(e_i, e_j) \mid e_i \mid e_j \mid nonce \mid MAC_{bill}$. $b(e_i, e_j)$ represents the bill; e_i and e_j are used to re-compute the bill for verification purposes; a nonce value is adopted to prevent a replay attack. A Message Authentication Code (i.e., MAC_{bill}) that covers the previous three fields is attached so that the receiver can check the integrity of the bill message. Then M_{bill} will be encrypted and transmitted through a secure channel. We let the bill exchange process follow a request-response pattern. When it is time to exchange bill message, the provider sends a request message that contains its own bill. Upon receiving it, the user first computes its bill based on e_i and e_j in the request message, and then replies to the provider with its own bill.

6. We offer two strategies to exchange the billing information. First, two meters can follow a constant time window so that bill messages are exchanged periodically. Second, the time window is optimized in terms of the system overhead (i.e., throughput in this case). We have formalized the TWM problem in Section 4.4.2. To adopt the optimized time window strategy, a meter needs to compute the time window length every time it receives a bill message.
7. After the bill message is received, user k is able to calculate the actual bill difference, which is the result of the user's bill after subtracting the provider's bill. Possible cases are discussed below:
    a. If the actual bill difference is greater than 0 and is less than the threshold b_0, the difference is minor and acceptable.
    b. If the difference exceeds b_0, there are four possibilities: first, the user manipulates the bill data and attempts to pay less than the amount he/she should pay; second, the provider manipulates the bill and attempts to charge more; third, both the provider and the user misbehave; fourth, both the provider and the user have no problem, and the cause is from outside (e.g., environment factors). The last case is a false alarm. No matter what the reason may be, the service is terminated, and further investigations will be initiated.
    c. However, if the actual difference is less than 0, there must be some problems with the meters because the provider is unable to provide less energy than the amount that the user has consumed. A report will be filed based on the incident.

The mutual inspection scheme is scalable and easy to implement. The scheme can be easily tailored to fit the three kinds of structures that we discussed earlier, because in nature the power service, no matter which structure it adopts, involves two parties (i.e., the provider and the user) while mutual inspection aims to provide accountability between the two parties. In addition, the implementation is feasible. The smart meter software or firmware is the only part that needs to be upgraded. The scheme is also scalable in terms of message overhead. Current AMI requires smart meters report reading value at regular intervals. Our scheme keeps it but the message quantity doubled since we adopt a request-response process. Every time a new meter joins, message overhead increases, but the overall complexity is linear.

4.5.3 Security Analysis

Confidentiality: Confidentiality can be provided by both symmetric and asymmetric encryption. During authentication, data is encrypted by the public key of the other end. Once authentication is accomplished, a session key is negotiated to establish a secure channel, through which every message is encrypted.

Integrity: Integrity is ensured by the MAC code attached with each bill message. In addition, the original entries are included in the message in case the other side needs to re-compute the bill. In the real world, a hash function (e.g., SHA-1) can be used as its implementation.

Accountability: Accountability can be ensured when 1) misbehavior can be detected, and 2) any misbehavior can be traced back to a responsible entity. In our context, it means that when excessive bill difference is detected, it should be able to determine which party (provider or user) misbehaved. Currently, the mutual inspection strategy can only achieve the first goal, and leave the second goal to further investigation.

Spoofing attack: A user's identity is bound with its public key certificate, which is signed and issued by a CA. If the CA is trustworthy, then certificates cannot be forged. Therefore, it is unlikely to impersonate other users without breaking the CA.

Replay attack: Replay attack can be prevented by adopting unique nonce value, which is in nature a pseudo random number. Each nonce is only for one-time use, making replay attack ineffective.

Man-In-The-Middle (MITM) attack: MITM attack can be performed when an attacker can take over the communication without being detected by the two parties. In our protocol, an MITM attacker has no means to break into the communication in the process of both handshake (authentication) and bill exchange. Since messages are encrypted by public keys (during handshake) or session keys (during bill exchange), an attacker can only capture the cipher text rather than become a middle man by manipulating the message.

4.6 Evaluation

In this section, in order to evaluate the method, we make the problem more concrete with stronger assumptions.

4.6.1 Power Demand Function

To define the power demand function, we assume that each smart appliance has two modes, i.e., on and off. Once an appliance is on, the capacity is fixed. Power demand function $P(t)$ gives the amount of service of a user at time *t*. Obviously, $P(t)$ depends on the modes of all electronic appliances. $P(t)$ can be calculated as $P(t)=\sum_{k=1}^{K} d_k(t)\cdot p_k$, where $d_k(t)$ = 1 if appliance *k* is on and 0 otherwise, and p_k stands for the capacity of appliance *k*. There are K appliances in total. Based on the assumption, we know that $P(t)$ is a piecewise function of time *t*.

4.6.2 Bill Difference Function

The bill difference function defines how time window Δt affects the bill difference. The price function $M(t)$ could be either a continuous function or a piecewise function.

- *Case 1*: $M(t)$ is a continuous function. In this case, $M(t)$ is continuous, and $P(t)$, as what we have assumed, is a piecewise function. The bill difference function can be transformed to $b(t_0,\Delta t)=\frac{R}{V^2}\left(\sum_{i=0}^{u}\left(P(t_i)^2\int_{t_i}^{t_{i+1}}M(t)dt\right)\right)+\alpha(t_0,\Delta t)$, in which $t_{u+1}=t_0+\Delta t$. This means there are $(u+1)$ segments of $P(t)$ in total between $[\,t_0,\Delta t+t_0\,]$. Since $M(t)$ is continuous, it is difficult to solve the constraint $b(t_0,\Delta t)\le b_0$. However, the Newton's method can help us find an approximate solution.
- $M(t)$ is a piecewise function. In this case, both $M(t)$ and $P(t)$ are piecewise functions. Then the previous equation can be transformed to $b(t_0,\Delta t)=\frac{R}{V^2}\left(\sum_{i=0}^{q}\left(\left(t_{i+1}-t_i\right)\cdot M(t_i)\cdot P(t_i)^2\right)\right)+\alpha(t_0,\Delta t)$, in which $t_{q+1}=t_0+\Delta t$, meaning that there are $(q+1)$ segments in total within $[\,t_0,\Delta t+t_0\,]$. Since both $M(t)$ and $P(t)$ are known piecewise functions, we can solve the constraint $b(t_0,\Delta t)\le b_0$.

4.6.3 Throughput Calculation

The notations (see Table 4.1) we use in this chapter are consistent with the ones in [59].

Table 4.1 Notations

Notation	Description	Notation	Description
T_{slot}	A slot time	T_{H_DATA}	Transmission time of MAC overhead
T_{SIFS}	SIFS time	T_{ACK}	ACK transmission time
T_{DIFS}	DIFS time	L_{DATA}	Payload size in bytes
$CW_{\min}$	Minimum back-off window size	T_{DATA}	Transmission time for payload
T_P	Transmission time of the physical preamble	T_{SYM}	Transmission time for a symbol
T_{PHY}	Transmission time of the physical header	τ	Propagation delay
L_{H_DATA}	MAC overhead in bytes, i.e., 14 bytes	R_{DATA}	Data rate
L_{ACK}	ACK size in bytes, i.e., 14 bytes	R_{ACK}	Control rate

In our problem, the actual data traffic depends on the time window Δt. When Δt is large, the data traffic is very low, and vice versa. In the extreme case, when Δt approaches 0, the data traffic can achieve full speed. According to [59], we can determine the actual throughput in IEEE 802.11a as follows,

$$H(\Delta t) = \frac{8L_{DATA}}{\Delta t + T_{D\_DATA} + T_{D\_ACK} + 2\tau + T_{DIFS} + T_{SIFS} + \overline{CW}} \tag{4.1}$$

in which $\overline{CW}$ is the average back-off time, given by $\overline{CW} = \left(CW_{\min} T_{slot}\right)/2$, and we also have,

$$T_{D\_DATA} = T_P + T_{PHY} + T_{SYM} \cdot Ceiling\left(\frac{16+6+8L_{H\_DATA}+8L_{DATA}}{N_{DBPS}}\right) \tag{4.2}$$

$$T_{D\_ACK} = T_P + T_{PHY} + T_{SYM} \cdot Ceiling\left(\frac{16+6+8L_{ACK}}{N_{DBPS}}\right) \tag{4.3}$$

Combining equations (4.1) - (4.3), we can obtain $H(\Delta t)$ and then solve constraint $H(\Delta t) > H_0$ under the assumption that there is no limitation on the data transmission rate. In the rest of this subsection, we analyze the problem using a numerical method. We provide a case study to verify the applicability of our approach. Some parameters in IEEE 802.11a are given in Table 4.2.

Table 4.2 Parameters of IEEE 802.11 a

Parameter	Value	Parameter	Value
T_{slot}	9 μs	T_{SIFS}	16 μs
τ	1 μs	$CW_{\min}$	15
T_P	16 μs	T_{PHY}	4 μs
T_{DIFS}	34 μs	T_{SYM}	4 μs
Data rate	54 Mbps	N_{DBPS}	216 bits

4.6.4 Numerical Evaluation

To better understand the performance, we assume that there is no limitation on the data transmission rate, and we also assume that the Smart Grid employs IEEE 802.11 as the communication standard. Based on paper [59], there is a throughput limit in IEEE 802.11 standards. We only consider an ideal one-hop and one-way communication, in which only two nodes are involved, i.e., the sender and the receiver. In our problem, the actual data traffic depends on the time window Δt. When Δt is large, the data traffic is very low, and vice versa. In the extreme case, when Δt approaches 0, the data traffic can achieve full speed.

Based on physics, wire resistance R can be calculated by $R = (\rho \cdot L)/S$, where ρ stands for the resistivity, L is the wire length, and *S* represents the cross sectional area. Given that most power wires are made of copper, we let $\rho = 3.06 \times 10^{-7}$ (Ωm). Wire length varies. As a case

study, we let $L = 100$ (m), and $S = 1.6 \times 10^{-5}$ (m^2). Based on the parameter setting, we have R = 1.9125 (Ω). The voltage is 110 v, which is the standard voltage in North America.

Since we have assumed that $P(t)$ is a piecewise function, a test case function of $P(t)$ is given as follows:

$$P(t) = \begin{cases} 500 \text{ (w)} & \text{t=0} \\ 700 & \text{t=8} \\ 2000 & \text{t=13} \\ 1300 & \text{t=22} \end{cases}$$

In this case, we consider $M(t)$ as a piecewise function (the case of M(t) being continuous has similar results), and repeat every 24 hours. We let $M(t)$ be:

$$M(t) = \begin{cases} 0.2 \text{ (\$/kwh)} & \text{t=0} \\ 0.3 & \text{t=10} \\ 0.4 & \text{t=18} \end{cases}$$

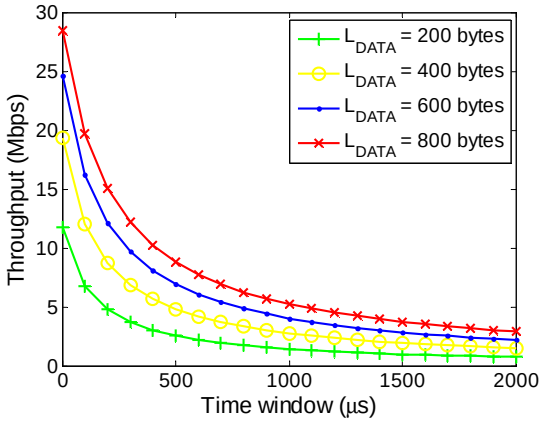
If we let $t_0 = 0$ then the bill difference function can be determined. In this specific case, we describe the following bill difference function as follows:

$$b(0, \Delta t) = \begin{cases} 7.9 \times \Delta t & 0 \le \Delta \text{t} < 8 \\ 15.484 \times \Delta t - 60.672 & 8 \le \Delta \text{t} < 10 \\ 23.226 \times \Delta t - 138.092 & 10 \le \Delta \text{t} < 13 \\ 189.6 \times \Delta t - 2301 & 13 \le \Delta \text{t} < 18 \\ 252.8 \times \Delta t - 3438.6 & 18 \le \Delta \text{t} < 22 \\ 106.8 \times \Delta t - 226.73 & 22 \le \Delta \text{t} < 24 \end{cases}$$

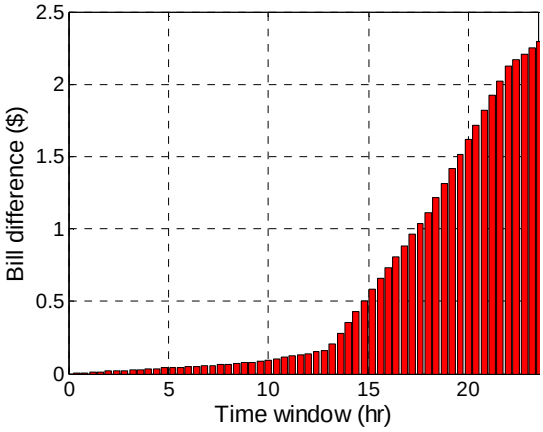
In Fig. 4.2-a, when the time window Δt increases from 0 to 500 μs, the throughput keeps decreasing. This is reasonable since the larger the time window is, the lower the communication frequency is, and this decreases the throughput. We can also observe that when the time window is fixed, throughput increases when the payload data becomes larger.

Fig. 4.2-b shows how time window Δt influences the bill difference. In our case, when $t_0 = 0$, the bill difference will become larger with the increase of Δt. Based on this result, we can determine the maximum Δt with a given bill difference threshold.

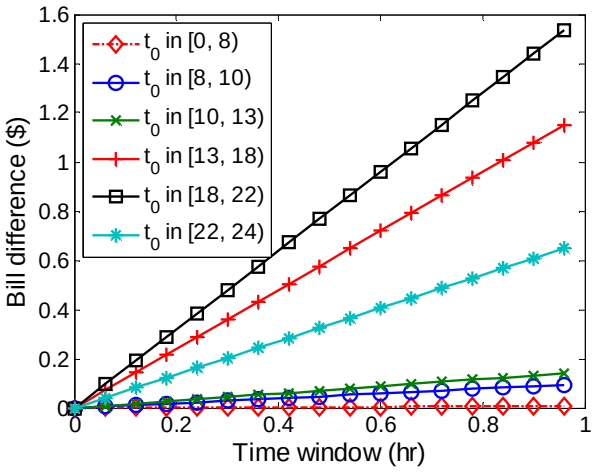
Fig. 4.2-c shows how different t_0 changes the relation between the time window and bill difference. We can observe that t_0 is a key factor for determining the actual bill difference, because both electricity price and user demand are constantly changing. Once any of those changes, the way to calculate the bill difference also changes. This illustrates that every time a dispute is resolved, a new Δt should be computed based on the current status.



(a) Time window and throughput



(b) Time window and bill difference



(c) Time window and bill difference when t_0 lies in different intervals

Fig. 4.2 Numeric results

4.6.4 Simulation Results

The two strategies being compared are the constant time window (TW) and the optimized TW. The former strategy adopts a constant time window for message exchange. The latter, offers an optimal time window to reduce the message overhead. The mutual inspection approach is applicable to the Smart Grid regardless of its architecture model. Therefore, in the simulation, we only focus on one smart grid structure (i.e., the p2p architecture) as the evaluation environment.

We select four metrics to evaluate the scheme. The first is P-Accountability. This is a metric to evaluate the degree of accountability. In this context, we define P-Accountability = (the number of detected malicious meters) / (the number of total malicious meters). The second is average detection time that measures how long it takes to detect anomaly, since it is important to know how promptly the system responds to misbehavior. The third is the false alarm rate, since it could be regular power loss or other accidental causes that trigger the alarm. The simulation attempts to discover how dispute threshold would affect the false alarm. Intuitively, a higher threshold implies a lower the false alarm rate. On the other hand, P-Accountability will be

reduced as well since the average detection time will be prolonged. The fourth is message overhead.

The parameters we use are 1) the Smart Grid scale (i.e., the number of independent users), 2) the ratio of malicious meters, and 3) the dispute threshold. We also assume that each independent user has a fully functional smart meter that is able to measure all input/output electricity amounts in real time.

Fig. 4.3-a describes the way the smart grid scale affects average detection time. In this experiment, we set the malicious node ratio to 0.1 and the dispute threshold to $1. We have compared the optimized TW method with the constant TW method. The result shows that the Smart Grid scale does not affect the average detection time regardless of the time window method we adopt. This is explanatory since the mutual inspection mechanism enables the provider and the user to inspect each other, and the scale does not add much complexity. Additionally, it shows that the optimized time window method can achieve fair performance in terms of average detection time. Although it is not as good as the case when the time window is fixed at 5 *ms*, we can show that it generates less overhead in later experiments.

Fig. 4.3-b shows how malicious meter ratio relates to the false alarm rate. A false alarm means that a user accuses another user by mistake. A false alarm is possible because the dispute threshold cannot completely satisfy every case. For example, if the threshold is too high, real malicious meters may escape detection; if, however, the threshold is too low, the bill difference caused by regular variance could be detected, and this is where the false alarm comes from. From this figure, we can observe that the threshold and the malicious meter ratio are two key factors that affect the false alarm rate. When the malicious meter ratio is higher, there are fewer false alarms.

P-Accountability (shown in Fig. 4.3-c) is defined as the ratio of detected malicious meters number and the number of all malicious meters. The major parameter that affects P-Accountability is the bill difference threshold. We can observe that when the threshold increases from \$0.1 to \$0.7, P-Accountabilty decreases. This means that a higher threshold allows some malicious meters to escape. This is similar to the former experiment. In nature, P-Accountability and false alarm partially work against each other. They are used to evaluate the system performance from two aspects.

Overhead is another performance issue. In the experiment, we use the message overhead as the metric to evaluate the optimized TW method and the constant TW method. It can be observed that the optimized TW method performs rather well when compared to the constant time window method. It introduces relatively low overhead while still maintaining low average detection time (Fig. 4.3-d).

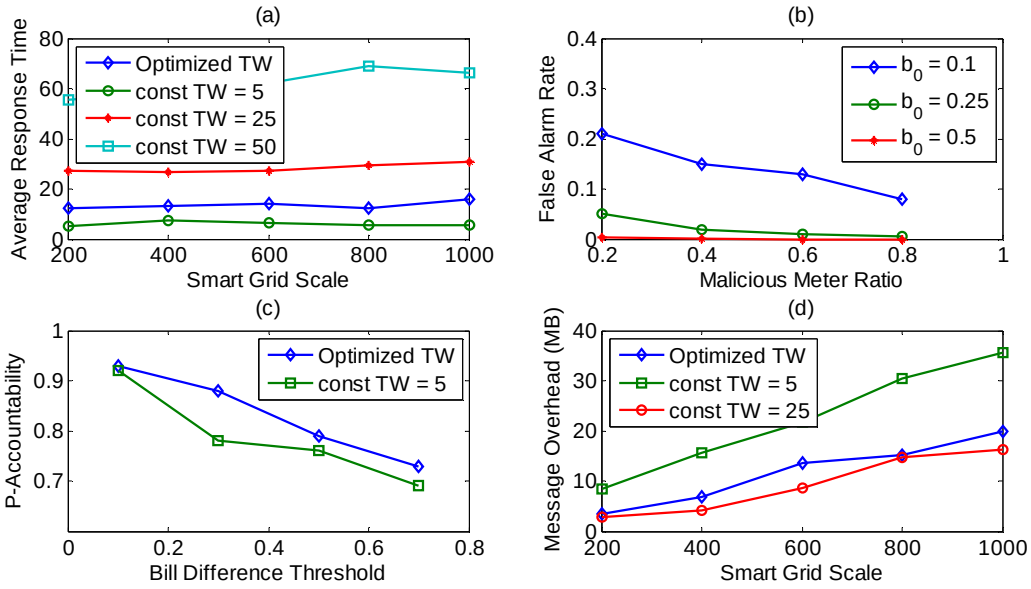


Fig. 4.3 Simulation results

4.7 Conclusion

The lack of non-repudiation is a major barrier of the trustworthy Smart Grid. In current power systems, bills are generated based on the amount of service consumed by users. However, meter readings may not be trustworthy due to malicious behaviors (e.g., energy theft) or external attacks. The root cause is that power providers have no means to obtain the reading other than receiving it from the users. To resolve this issue, we proposed a mutual inspection strategy, which enables non-repudiation on meter readings for the Smart Grid in the neighborhood area network. The goal of our scheme is to discover problematic meters that report inaccurate reading value.

CHAPTER 5

EXPLORING MALICIOUS METER INSPECTION IN THE NEIGHBORHOOD AREA SMART GRID

5.1 Introduction

In the last chapter, we have proposed a mutual inspection scheme, which installs a redundant meter on the provider end to monitor each user. This means that for each individual power line, there is one smart meter on each end. The difference between the two readings should be within a certain range. Otherwise, a dispute will arise, indicating that the smart meter may be under an attack.

One limitation of the mutual inspection scheme is its cost. Adding a redundant meter for each user will increase the expense of hardware and management. The cost issue is magnified in the metropolitan area (e.g., the New York City, Beijing, Tokyo, etc), since in big cities there are a plenty of apartment buildings assembling numerous apartment units. According to the mutual inspection scheme, the power provider should install a redundant smart meter (referred to as an **inspector**) on the provider's end of the power line, thus the number of inspectors is equal to the number of users. In this chapter, we consider a situation that the number of inspectors is far less than the number of smart meters to be checked.

Our main contributions are summarized as follows: Firstly, we propose the Malicious Meter Inspection (MMI) problem. To our best knowledge, our work is the first study addressing the MMI problem for the Smart Grid. The goal is to reduce inspection cost and still maintain

efficient inspection. Secondly, we propose a suite of algorithms to solve the MMI problem for both the static and dynamic cases. The algorithms are based upon an inspection tree. We also find out that the binary-tree-based inspection is not necessarily better than the naïve scanning approach. As a result, we propose an adaptive-tree-based inspection scheme that can collect heuristic information and adjust the inspection strategy in real time. We analyze and give the performance bounds for each algorithm. Finally, we give both numerical and experimental results for the discussed algorithms. It turns out that the adaptive tree approach is a good choice in general circumstances.

5.2 Related Work

Bandim *et al*. [65] employ a central observer meter, which measures the overall energy consumption of a group of N end users. By taking N samples at different moments, there will be N equations with N unknown constants (e.g., the accuracy coefficients), thus the equation set is solvable. We argue that the assumption that an accuracy coefficient is constant may not hold because it is limited by the attack model. If an accuracy coefficient is a variable, which is highly possible when an attacker changes its strategy, then the method would not work.

In this work, we provide a comprehensive solution that can defend against multiple attack models with undeniable evidence to identify the malicious meters.

5.3 Malicious Meter Inspection Problem

Notations for this chapter are given in Table 5.1. In general, we use a capital letter to name a set and a corresponding lower case letter to denote an element in that set.

The Smart Grid model: Consider an apartment building with n users (i.e., each apartment represents a user), which compose a user set U. Each user has one meter installed to

measure the amount of service. In this chapter, the terms, 'user' and 'meter,' are interchangeable since they represent the same thing. In our setting, there is a head inspector, which is also a smart meter, to monitor the entire building all the time. In reality, it is reasonable and affordable to have a central monitor for each building. If there is any meter reading error in the building, the head inspector can detect it by comparing its own reading with the summation of all reported readings. If the difference is larger than a threshold, the head inspector will trigger an alarm.

Table 5.1 Notations for MMI

Notation	*Description*
U	A set of users.
n	$\|U\|$:= n, the total number of users (smart meters).
Q	A set of malicious users. We also have $Q \subseteq U$.
m	$\|Q\|$:= m, the number of malicious users.
N_S	The number of steps required to identify a complete Q.
N_I	The number of inspectors.
$Perm_U$	One permutation of U's elements, representing an instance of the inspection tree. Let $Perm_U := [v_1 v_2 ... v_n]$ in which v_i is binary value. $v_i = 0$ means user u_i is good, and $v_i = 1$ means it is malicious.
γ	Malicious meter ratio, given by $\gamma = m/n$

Attack model: McLaughlin *et al*. [53] have investigated the attack models of energy theft with an attack tree, which summarizes the approaches of energy theft. Based upon their security analysis and case study on an experimental test-bed, it is practically viable to tamper with the reading of a smart meter. In this chapter, however, we need to point out that although energy theft may be the main incentive of a meter attack, it is not the sole motive. We consider two kinds of attack models, which cover the ones discussed in [53]. In our setting, a meter can be attacked by external adversaries or be manipulated by a malicious user. For the case of energy theft, it is the user's intent to compromise the meter. Apart from that, external attackers may

increase the reading values so that victims will have excessive power bills. Therefore, it is possible to bypass the head inspector. For example, a malicious user that controls multiple meters may decrease the reading for his home and increase other meters' reading. The total demand of these meters may remain unchanged, and head inspector is unable to detect any anomaly. It is not hard to avoid this attack by randomly monitoring a subset of users during a period of time. Thus, the probability of successfully bypassing the head inspector is very low. We need to point out that this attack may be specific to our model for the Smart Grid.

5.3.1 Problem Statement

For the purpose of inspection, we assume that a device called the inspector box, which is located between the head inspector and end meters, exists. Such an inspector box is capable of managing three things: 1) each power line will be connected to an end user at all times to prevent an outage; 2) the inspector box contains a number of inspectors, and it is effortless to add or remove inspectors; 3) it is viable to assign any user combination with any number to one inspector, and this can be done manually or automatically. Fig. 5.1 describes a possible design for the inspector box with one inspector.

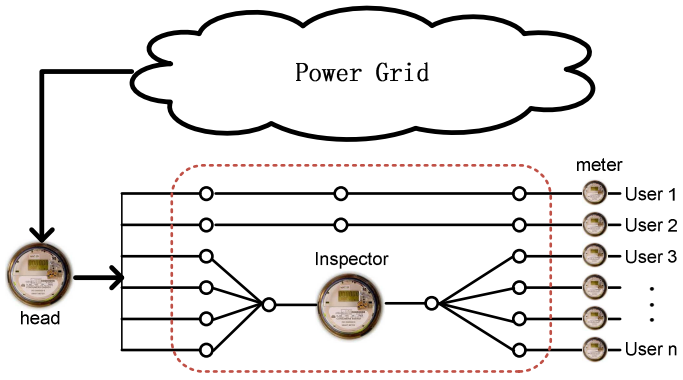


Fig. 5.1 An inspector box located outside each apartment building

The head inspector is able to detect reading anomalies existing in the building, although it is still unclear which ones are malicious. A definite answer can be given only if a test is carried out by one inspector on one user meter. With one inspector, it takes n tests to identify a complete set, Q, under a one-by-one test scheme. To reduce the number of tests, we can either 1) increase the number of inspectors because multiple inspectors can do a parallel test, or 2) use a better test scheme. In this chapter, we follow a multi-step strategy to identify the compromised meter(s) with a limited number of inspectors.

There are two parameters for the method evaluation: (1) The time spent on identifying the malicious meters (i.e., set Q). In this problem, we abstract the time as the number of steps (denoted as N_S). (2) The number of inspectors (denoted as N_I), excluding the head inspector.

Malicious Meter Inspection Problem (MMI): in the smart grid model, given N_I inspectors, the objective is to minimize the number of steps (i.e., N_S) needed to identify a complete Q.

Based upon the variability of Q, we will discuss two types of inspection: *static* inspection and *dynamic* inspection. Static inspection handles the case that Q does not change during the inspection process, while dynamic inspection considers a changeable Q over the inspection process. In this chapter, we investigate the *MMI* problem for both static and dynamic inspection.

5.3.2 MMI and Group Testing

To some extent, the *MMI* problem is similar to the group testing problem [64], which was invented by R. Dorfman in World War II to facilitate the procedure of identifying syphilis in blood samples from millions of draftees. Du *et al*. [63] have provided a survey that summarized the group testing problem and its variations. A plenty of prior studies have been done regarding the algorithm improvements, problem variations, and special case discussions. We discover that

the MMI problem and the group testing problem have certain overlaps in definition. However, the distinction is also obvious. In conventional group testing problems, the defectives will never change once the item set is given. Nevertheless, in the MMI problem, the number of bad meters may increase or reduce during inspection, which is more challenging to identify all of them within a short period of time. The MMI problem can be regarded as a variation of group testing with dynamic feature. Furthermore, our proposed solutions are different from the existing group testing methods.

5.4 Static Inspection

For static inspection, set Q does not change. In reality, Q could change in the long run. However, in a short period, Q can be regarded to be static. Therefore, we first study the case of static inspection.

5.4.1 Scanning Approach – a Brute-force Strategy

When set Q is static, the naïve solution is to scan every meter iteratively. Q will be eventually identified after all of the meters are checked one by one. There are two extreme cases. 1) $N_S = 1$ and $N_I = n$. In this case, we deploy n inspectors, which do the inspection job in parallel so that each user meter will be checked in one step; 2) $N_S = n$ and $N_I = 1$. In this case, we use one inspector to test a single end meter at one step so that it takes n steps to test all users.

The advantage of scanning is that the step complexity, N_S , is bounded by the number of users (i.e., n). In the real world, m is unknown prior to inspection. If m is far less than n, scanning may not be efficient because it spends a lot of time checking the good meters, which wastes time and inspectors. By taking advantage of the Smart Grid property, we have designed a tree-based inspection scheme, which employs a tree as a logic structure to identify the compromised meters.

5.4.2 Binary-tree-based Inspection

With the assistance of an inspector box, we can model the inspection process as a binary tree *T*, in which each node represents one time of inspection (i.e., one step). In addition, each leaf node connects with a smart meter being checked. With the binary tree, we can show that when an inspector is deployed in an internal node, it covers all meters in its subtrees. For example, in Fig. 5.3, if node *b* is an inspector, it is able to inspect meters 1, 2, 3, and 4, which all belong to the subtrees of *b*. An internal node can tell whether there is any anomaly in the subtree, but it does not know exactly which one is bad, meaning that the inspection should continue. On the other hand, although an inspector in a leaf node can only examine one meter, it gives evidence showing a meter's real status. In Fig. 5.3 and other related figures where the inspection tree is applied, symbol 'X' means an anomaly is detected (we call the node is **dirty**), and 'O' means the meter is working correctly (we call the node is **clean**). Additionally, the light blue circle means the node is actually checked, while the dark blue circle means the node does not need to be checked because its parent indicates its status.

Table 5.2 Probing

Algorithm: Probing
Input: a node in binary tree *T*
Output: 'dirty' or 'clean'
probe (*node*):
    Add an inspector at *node x*;
    if ($\left| R(node) - \sum_{u_i \in CM(node)} R(u_i) \right| > \sum_{u_i \in CM(node)} \delta(u_i)$)
        return 'dirty';
    else return 'clean';

A node's status can be determined by probing, which is given in Table 5.2. A probing operation describes how an inspector checks the correctness of meters. In the algorithm, function

$R(x)$ returns the reading at node x, and function $CM(x)$ returns the set containing all meters in the subtrees of node x. δ is a threshold determined by the power loss during transmission. Specifically, $\delta(u_i)$ denotes the threshold for user u_i. According to the electricity knowledge, the amount of lost electricity depends on the wire resistance, the transmission voltage, and the power demand of user. In addition, the wire resistance depends on the cable material, cable length, and environment factors like temperature. Since some factors (e.g., the power demand and cable length) vary for different users, each user i will have a threshold which is also a variable. A concrete model of the threshold will not be discussed since it is not in the scope of this chapter. If the difference between *R(x)* and the sum of reported readings from user meters is greater than the sum of the threshold values, then the node is considered to be *dirty*. The execution of probing will be logged in a tamper-evident way as evidence, which will be presented during the process of forensics.

1)Building a binary inspection tree

Given a user set U, we build a binary tree with the n meters that are randomly picked as leaf nodes. For any n, we have $2^i \le n < 2^{i+1}$ ($i \ge 0$). When $n = 2^i$, the binary tree is complete. Otherwise, the tree is incomplete. The process of building a tree is described as follows:

- If $n = 2^i$, then a complete binary tree is built with height $\log_2(n+1)$. The tree has n leaf nodes, which correspond to the n meters in U.
- If $2^i < n < 2^{i+1}$, the binary tree is built to be incomplete. We first build a complete binary tree with 2^i leaf nodes, then the rest ($n - 2^i$) nodes can be added to the tree one by one starting from the leftmost leaf node. In order to maintain the property that each leaf node represents a meter, we introduce a PullDown operation (as

shown in Fig. 5.2) when a new meter is added. After the tree is built, the status of a leaf node can be assigned from left to right.

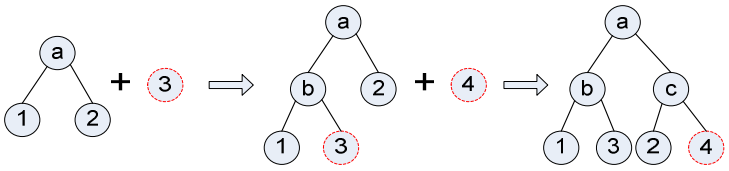


Fig. 5.2 The PullDown operation

*2)**Case 1**: m = 1, and N_I = 1*

This case may not be realistic since it is difficult to determine how many meters are dirty prior to inspection. However, it presents the basic idea. In Fig. 5.3, root *a* is detected as a dirty node due to meter 2. The inspection is performed in a top-down pattern.

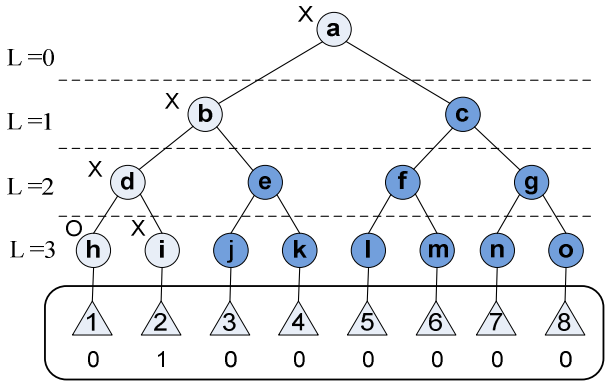


Fig. 5.3 An example of binary inspection with $Perm_U$= [01000000], *m*=1, and N_I = 1

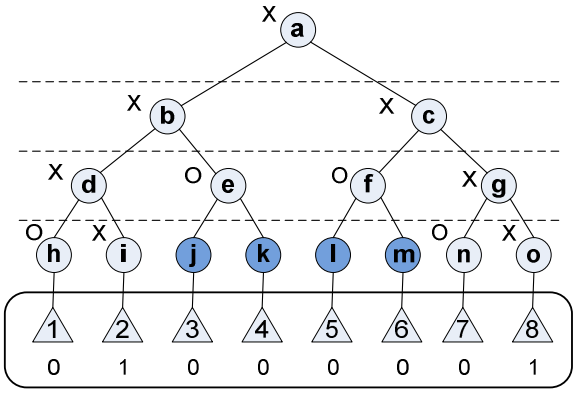


Fig. 5.4 $Perm_U$ = [01000001], *m* = 2 and N_I = 1

Based upon Algorithm BI_basic (shown in Table 5.3), we can determine the upper bound and lower bound of N_S. Since the height of the binary tree is $\lceil \log_2 n \rceil + 1$, we have $\lceil \log_2 n \rceil + 1 \leq N_S \leq 2 \cdot \lceil \log_2 n \rceil + 1$. The lower bound and upper bound can be achieved when the dirty node is the leftmost and the rightmost one, respectively. Note that the purpose of this simple case is to demonstrate the idea of a tree-based inspection. We will discuss more general cases later.

Table 5.3 BI_basic and BI_v1

Algorithm: basic binary-inspection (BI_basic) when $N_I = 1$ and m = 1	**Algorithm**: Binary-inspection Version 1 (BI_v1) when $N_I = 1$ and $m \geq 1$

Input: binary tree *root*
Output: the dirty node
Initial : *node := root*
BI_basic(*node*):
if (*node* == null)
return null;
if (probe(*node*) == 'dirty')
if (*node* is leaf)
return *node;*
// *node* is an internal node
temp := BI_basic(*node*.lchild);
if (temp == null);
return BI_basic(*node*.rchild);
else
return temp;
return null;
	Input: binary tree *root*
	Output: malicious meter set *Q*
	Initial: $Q := \varnothing$, *node* := root
	BI_v1(*node*, *Q*):
	if (node == null)
	return;
	if (probe(*node*) == 'dirty')
	if (*node* is leaf)
	$Q := Q \cup \{node\}$
	return;
	else // internal node
	BI_v1(*node*.lchild, *Q*);
	BI_v1(*node*.rchild, *Q*);
	else
	return;

Based upon Algorithm BI_basic, we can determine the upper bound and lower bound of N_S. Since the height of the binary tree is $\lceil \log_2 n \rceil + 1$, we have $\lceil \log_2 n \rceil + 1 \leq N_S \leq 2 \cdot \lceil \log_2 n \rceil + 1$. The lower bound and upper bound can be achieved when the dirty node is the leftmost and rightmost

one, respectively. Note that the purpose of this simple case is to demonstrate the idea of a tree-based inspection. We will discuss more general cases later.

Theory 5.1*: Algorithm BI_basic, is the optimal algorithm when m = 1 and* $N_I = 1$ *with* $N_S = o(\log_2 n)$ *if and only if each meter is equally likely to be a bad meter*.

Proof: Our previous analysis shows that for algorithm BI_basic, we have the bounds of N_s as follows: $\lceil \log_2 n \rceil + 1 \le N_S \le 2 \cdot \lceil \log_2 n \rceil + 1$. From the information theory point of view, the uncertainty of the problem is the entropy $H(x)$, which is equivalent to the number (*n*) of yes-or-no questions that are needed to accurately determine the outcome of a random event *x*. Based on entropy and information theory, we have: 1) $H(x) \le \log_2 n$ and 2) $H(x) = \log_2 n$ only if the random event is uniformly distributed (i.e., each meter is equally likely to be a bad meter). □

*3)**Case 2**: m is unknown,* $N_I = 1$

The only difference between case 1 and case 2 is that the number of dirty meters is unknown prior to inspection. Therefore, all children of a node need to be probed. Fig. 5.4 shows an example of this case. Note that node *e* does not need to be probed in case 1. The inspection process is described in Table 5.3 - BI_v1. The BI_v1 algorithm, a recursive algorithm, is not necessarily faster than the sequential scanning; for example, in Fig. 5.4, we have $N_S = 11$, which is more than *n* = 8.

Lemma 5.1*: For algorithm BI_v1, when* $N_I = 1$ *and Q is unknown (i.e.,* $m \ge 1$*), we have the following: The lower bound of* N_S *is* $|N_S|_{\min} = 2\lceil \log_2 n \rceil + 1$*, and the lower bound can be achieved only if* $m \le 2$ *and if m = 2, the two dirty leaf nodes must be siblings. The upper bound*

of N_S is $|N_S|_{\max} = 2n-1$, and the upper bound can be achieved only if $m \geq n/2$ and if for every pair of leaf nodes that are siblings, at least one of them must be dirty.

Proof: We first prove the lower bound. If $m = 1$, there will be one probing in the root node; for the rest of the nodes, there will be two nodes probed in each level of the tree until the leaf nodes are reached. Therefore, the total number of steps is $N_S = 2 \cdot \lceil \log_2 n \rceil + 1$. If m = 2 and if the two dirty nodes are siblings, then $N_S = 2 \cdot \lceil \log_2 n \rceil + 1$ still holds because no extra probe is needed. If m = 2 and if the two dirty nodes are not siblings, then the probing process will fork before it reaches any leaf nodes. The fork costs additional probing steps. If $m > 2$, more probing will be needed in each level. Therefore, the proof of the first part completes.

We then prove the upper bound. The upper bound is limited by the tree size because each node in the tree represents a probing. We use mathematic induction to show that no matter whether the tree is complete or not, the tree size is $2n-1$. When n = 1, there is only one node in the tree, and the tree size is $2n-1=1$. Assume that when $n=k$, the tree size is $2 \cdot k-1$. We need to show that when $n = k + 1$, the tree size is $2 \cdot n-1=2 \cdot k+1$. The fact is that every time we add a meter, we actually add 2 nodes to the tree (one leaf and one internal node) due to the *PullDown* operation (as shown in Fig. 5.2). Therefore, after adding one leaf node, the tree size is $2 \cdot k-1+2=2 \cdot k+1$, thus our claim still holds. Since each node could be one time of probing, it will need at most $2n-1$ times to find the complete Q. The upper bound can be achieved only if for every pair of leaf nodes that are siblings, at least one of them must be dirty. Let us consider that when all leaves are dirty, all the internal nodes need to be probed. If we change one meter of any two meters that are siblings to be clean, the total probing times (i.e., N_S) will not change. However, if we change both sibling meters to clean, then N_S will be reduced by two since these

two meters will skip probing because that their parent node is clean, and this decreases N_S. Therefore, the proof of the second part completes. □

From Lemma 5.1, we know that BI_v1 can be worse than the scanning approach in some situations. However, it is still not clear when and how BI_v1 will be better or worse. The key question we need to answer is "given that *m* out of *n* meters are dirty, how to determine N_S?"

We define two nodes as siblings if and only if they share a common parent. We also define six operations: one-round pairing, one-round depairing, one-level pairing, one-level depairing, whole-tree pairing, and whole-tree depairing. Both one-round pairing and one-round depairing work on nodes within the same level of the tree.

One-round pairing is a three-step process of finding two dirty nodes that are in the same level and that have clean siblings, switching one dirty node (and its subtree) with the other dirty node's clean sibling node (and its subtree) so that two dirty nodes become siblings, and two clean nodes becomes siblings, and adjusting the related non-leave nodes' states (dirty or clean) in higher levels to correct states accordingly. Note that when two nodes are switched, their subtrees are switched as well. *One-level pairing* is an iterative process of one-round pairing in one level until there are not nodes in the level to conduct any one-round pairing. Starting from the leave level, *whole-tree pairing* is an iterative process of one-level pairing, level by level up until one-level pairing is done for all levels. Note that whole-tree pairing may produce different results depending on the ways of choosing two dirty nodes to pair. For example, in Fig. 5.4, by switching node *h* and node *o*, we finish one-level pairing in the leave level.

One-round depairing, one-level depairing, and whole-tree depairing are reversed processes of one-round pairing, one-level pairing, and whole-tree pairing, respectively, although the results may not be the original settings depending on the ways of choosing two dirty nodes to

do one-round depair. In other words, *one-round depairing* is a three-step process of finding two sibling dirty nodes and two sibling clean nodes, switching one dirty node (and its subtree) with one clean node (and its subtree) among them so that each of the two dirty nodes has a clean sibling, and adjusting the related non-leave nodes' states (dirty or clean) in higher levels to correct states accordingly. *One-level depairing* is an iterative process of one-round depairing in one level until there are not nodes in the level to conduct any one-round depairing. Starting from the leave level, *whole-tree depairing* is an iterative process of one-level depairing, level by level up, until one-level pairing had been done for all levels.

Lemma 5.2: *For algorithm BI_v1, given an arbitrary binary inspection tree with m dirty leaves among n leaves, N_S for the result setting of whole-tree pairing reaches the minimum.*

Proof: Let $N_S = \sum_{i=1}^{h} N_S^i$, in which N_S^i denotes the number of probes in level *i*. Consider level *i*, we write the following $N_S = N_S^i + (N_S^1 + ... + N_S^{i-1} + N_S^{i+1} + ... N_S^h) = N_S^i + N_{rest}$, in which N_{rest} denotes the sum of number of probes for all levels but level *i*.

First, we study the effect of one-round pairing in level *i*. Before we do any one-round pairing, we need to find two dirty nodes with clean siblings. This means that all four nodes will be probed when BI_v1 is applied. After one-round pairing, two dirty nodes become siblings which still need probing, and two clean nodes become siblings which do not need probing any more. Therefore, each one-round pairing reduces the number of probes by 2 for the current level, i.e., $N_S^i := N_S^i - 2$. In addition, since after a one-round pairing, one dirty node in the immediate upper level will become clean, and the statuses for the rest nodes in that level remain unchanged. Therefore, the number of probes in the immediate upper level will never increase. For the same reason, the number of probes for all upper levels will never increase. Moreover, one-round

pairing in a higher level will not affect the number of probes in lower levels because the subtree structure of a node in the higher level will not change. In other words, the number of probes for each of the subtrees of nodes in the higher level remains the same. Therefore, after a time of one-round pairing in level *i*, we have $N_S^i := N_S^i - 2$, and N_{rest} will not increase.

Now, once we finish one-level pairing in level *i*, N_S^i is minimized because no more one-round pairing can be done in the level. Thus we have $N_S = |N_S^i|_{\min} + N_{rest}$. We start from the leave level, every time we finish pairing in a level, we go one level higher than the current level, and perform pairing iteratively, until we finish all the levels. When the whole-tree pairing is done, N_S^i for every *i* reaches the minimum, i.e., $N_S = \sum_{i=1}^{h} |N_S^i|_{\min}$. Therefore, N_S also reaches the minimum. □

The reason we choose a bottom-up process instead of a top-down process is for proof convenience. With bottom-up process, we first have $N_S = |N_S^h|_{\min} + \sum_{i=1}^{h-1} N_S^i$ after one-level pairing for level *h* (i.e., the leave level); then after one-level pairing for level *h-1*, we have $N_S = |N_S^h|_{\min} + |N_S^{h-1}|_{\min} + \sum_{i=1}^{h-2} N_S^i$, and $|N_S^h|_{\min}$ will not change due to the fact that high level pairing would not affect the number of probes in low levels. Therefore, we can obtain $N_S = \sum_{i=1}^{h} |N_S^i|_{\min}$ after the whole-tree pairing. However, if we choose a top-down process, we will first have $N_S = |N_S^1|_{\min} + \sum_{i=2}^{h} N_S^i$ after one-level pairing for level 1 (i.e., the top level), and then $N_S = |N_S^1|_{\min} + |N_S^2|_{\min} + \sum_{i=3}^{h} N_S^i$. However, after finish level 3, we can be sure that $|N_S^3|_{\min}$ is obtained, but the value of N_S^1 and N_S^2 may be further reduced because the number of dirty nodes in higher levels is reduced. Thus it is not easy to determine whether the numbers of probes for higher levels can

reach the minimum during the pairing process. In other words, equation $N_S = \sum_{i=1}^{h} |\, N_S^i \,|_{\min}$ is difficult to obtain when we do a top-down process for pairing.

For example, in Fig. 5.4, once the statuses of node *h* and node *o* are switched, node *n* and *o* will not be probed, and *h* will be probed regardless of its status; also, node *g* becomes clean, which may or may not reduce the number of probes, depending on the status of g's sibling. In this example, since *g*'s sibling *f* is clean, the fact that *g* becomes clean will also reduce N_S.

Lemma 5.3: *For algorithm BI_v1, given an arbitrary binary inspection tree m dirty leaves among n leaves, N_S for the result setting of whole-tree depairing reaches the maximum.*

Proof for Lemma 5.3 is omitted because the proof is similar to the proof of Lemma 5.2 except that every time a one-round depairing is finished, N_S will be increased at least by 2.

Lemma 5.4: *For algorithm BI_v1, given an arbitrary binary inspection tree with m dirty leaves among n leaves, we have* $|N_S|_{\min} = \begin{cases} 2\cdot(m+h)-2g(m)-1 & m=2k-1, k=1,2,...,n/2. \\ 2\cdot(m-1+h)-2g(m-1)-1 & m=2k, k=1,2,...,n/2. \end{cases}$'

in which h is the tree height, g(m) is the number of 1s in m's binary representation. We also have $|N_S|_{\max} = 2\cdot 2^{\lceil \log_2 m \rceil} - 1 + 2\cdot(\lceil \log_2 n \rceil - \lceil \log_2 m \rceil)\cdot m$, *in which m = 1, 2, …, n.*

Proof: 1). Given *P1* = [v_1,v_2, …v_m, v_{m+1}…, v_n] such that meters from v_1 to v_m are all dirty. *P1* is one possible result of a whole-tree pairing operation, because if a binary inspection tree is built based upon *P1*, there will be no one-round pairing that can be possibly done for the whole tree. Therefore, we can calculate $|N_S|_{\min}$ based on P1. Let function *f(m, h)* define the number of probes needed to find out all *m* dirty meters in a tree built from P1 with height *h*. *f(m, h)* can be expressed by recurrence: $f(m,h) = f(2^i, i+1) + f(m-2^i, i+1) + 2\cdot(h-i) - 3$, in which *m* ($2^i < m < 2^{i+1}$) is odd, and *h* is the tree height, i.e., $h = \lceil \log_2 n \rceil + 1$. The tree consists of three parts (see Fig.

5.5): $f(2^i, i+1)$ denotes part 1, which is a subtree full of dirty nodes; $f(m-2^i, i+1)$ denotes part 2; and $2\cdot(h-i)-3$ denotes part 3. We analyze the three parts separately. Note that part 1 is a binary tree full of dirty meters, thus its size is equal to the steps that are needed. We then have $f(2^i, i+1) = 2^{i+1}-1$ for part 1 where 2^i is the number of dirty meters in this subtree. Also there are m-2^i meters for part 2. Part 3 is the top part of inspection tree; thus before it reaches part 1 and part 2, it takes 1 probe (the root node) and 2 probes for other levels, if there is any. Therefore, it takes $1+2(h-(i+1)-1) = 2\cdot(h-i)-3$ steps for part 3. To calculate part 2, $f(m-2^i, i+1)$, we need to solve the recurrence. Based on the tree property, it is straightforward to obtain the following: $f(2^i, i+1) = 2^{i+1}-1$, $f(0,h) = 0$, $f(1,h) = 2h-1$, and $f(2x,h) = f(2x-1,h)$, x = 1, 2, 3, …, and i = 0, 1, 2, …, h-1. By solving the recurrence with the recurrence tree method [66], we have $f(m,h) = 2\cdot(m+h)-2g(m)-1$. Fig. 5.5 shows an example of *f(3,4)*.

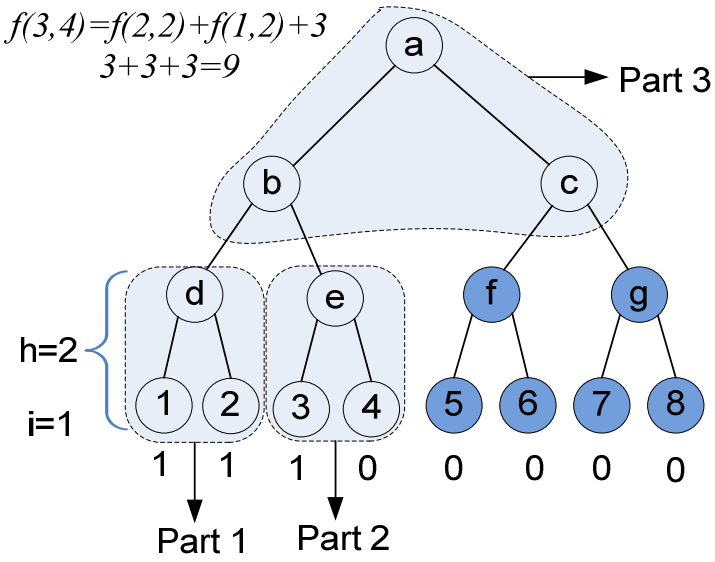


Fig. 5.5 An example of the best case of meter arrangement

2). If m is not a power of 2, let $2^i \le m < 2^{i+1}$, a general structure of the worst case is shown in Fig. 5.6, in which we can find a integer p such that the top p levels of the tree are full of

dirty node, and the (*p+1*)th level has exactly *m* nodes. We will first show that this structure is the result of a whole-tree depairing operation. In Fig. 5.6, since the top *p* levels are full of dirty nodes, no more one-level depairing can be done in the top *p* levels; also, we can ensure that every node in level *p* has at least one dirty child, which means there is no way to do one-round depairing in the (*p+1*)th level. In addition, each node in the (*p+1*)th level has only one dirty leaf in its subtree, which means no one-round depairing operation can be done in their subtrees. Therefore, this structure can be the result of a whole-tree depairing. Now we can calculate $|N_S|_{max}$ based on the structure. For the top p levels, the number of probes is $2 \cdot 2^p - 1$; for the rest *q* levels, each level will be probed by 2*m* times. Therefore, we have $N_S = 2 \cdot 2^p - 1 + 2 \cdot q \cdot m$, in which $p + q = \lceil \log_2 n \rceil + 1$, $p = \lceil \log_2 m \rceil + 1$. Combining the former equations, we have $N_S = 2 \cdot 2^{\lceil \log_2 m \rceil} - 1 + 2 \cdot (\lceil \log_2 n \rceil - \lceil \log_2 m \rceil) \cdot m$. □

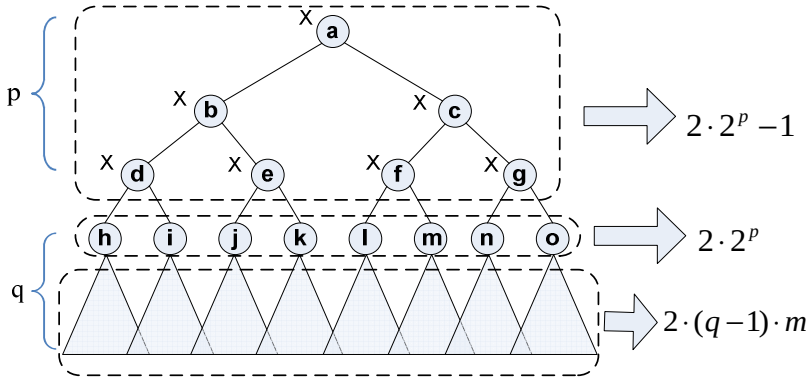
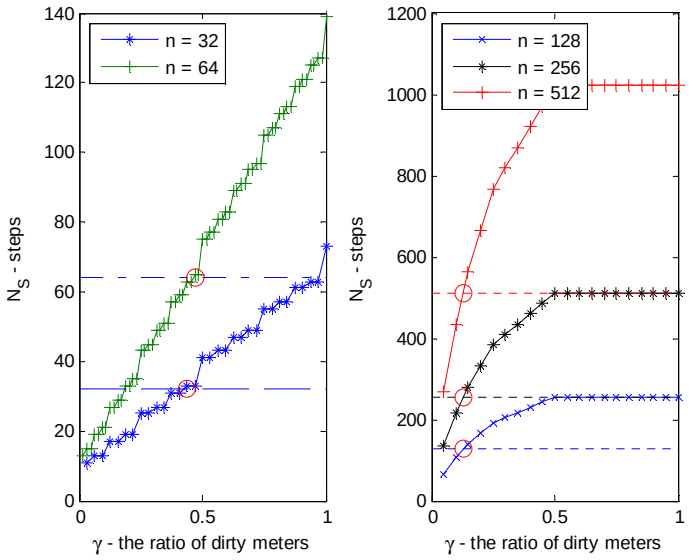


Fig. 5.6 The worst case of meter arrangement

We compare BI_v1 and scanning in Fig. 5.7. For the best case (i.e., lower bound), BI_v1 is better than scanning when the dirty node ratio is less than 0.5. For the worst case (i.e., upper bound), BI_v1 is better than scanning when the dirty node ratio is less than 0.13.



(a) lower bound **(b) upper bound**

Fig. 5.7 Numeric results of BI_v1

*4)**Case 3**: m is unknown, $N_I > 1$*

When there are multiple inspectors, it is straightforward to distribute workload by assigning the next node to be checked to the next available inspector. If we let N_S be a function of N_I, then the exact number of steps is $N_S(N_I) = \lfloor N_S(1)/N_I \rfloor + N_S(1) \bmod N_I$.

5.4.3 BI_v2: an Improvement of BI_v1

The BI_v1 algorithm can be improved by reducing the internal node checks. BI_v2 is described as follows: if a node is probed dirty and if *node.lChild* is probed to be clean, then *node.rChild* is definitely dirty and can be skipped to save steps. For example, in Fig. 5.4, after node *c* and node *f* are probed, we can be sure that *g* is dirty so that it can be skipped, and the next node to be probed is node *n*. Then, for the same reason, we can determine that node *o* is dirty without probing it. BI_V2 is given in Table 5.4 as follows:

Table 5.4 Binary inspection version 2 (BI_v2) when N_I=1 and *m* is unknown

```
Algorithm: BI_v2
Input: binary tree root
Output: malicious meter set Q
Initial: Q := ∅ , node := root
BI_v2(node, Q):
    if (node == null)
        return;
    if (probe(node) == 'dirty')
        if (node is leaf)
            Q := Q ∪ {node}
            return;
        else          // internal node
            if (probe(node.lchild) == 'clean') // skip node.rchild
                BI_v2(node.rchild.lchild, Q);
                BI_v2(node.rchild.rchild, Q);
            else
                BI_v2(node.lchild, Q);
                BI_v2(node.rchild, Q);
    else
        return;
```

5.4.4 Adaptive Binary Tree Inspection

Based upon the previous analysis, we determine that the choice of inspection scheme is dependent upon the dirty meter ratio and dirty meter arrangement, and both are unknown prior to inspection. However, we can collect some heuristic information during inspection. The two parameters (i.e., the ratio of dirty meters and its arrangement) can be obtained on line, and the inspection process can be adjusted adaptively to tune the performance.

1) Jumping

Jumping is a strategy used to reduce probing times. When the dirty meter ratio is high, scanning may be preferable. By skipping some internal nodes to directly probe their children in a lower level, jumping provides a mixing form of the binary tree inspection and scanning. Jumping range means the number of levels skipped by jumping, which is termed by D-Jumping, and D denotes the range of jumping. For example, 1-Jumping means it skips one level in the tree. To facilitate the Jumping operation, the node structure is modified so that each node holds two more

pointers (named *parent* and *right*): one points to its parent, and another points to its immediate neighbor on the right side.

2)A heuristic approach

To actually apply the jumping strategy, we develop the Adaptive Tree Inspection (ATI) algorithm. The ATI algorithm is based upon a claim: in a binary inspection tree, sub-trees in the same level are similar in terms of the dirty meter ratio and arrangement. The claim holds because the inspection tree is built in a random manner (i.e., each leaf node is selected randomly during the tree-building process) so that the probability that dirty meters are compactly arranged is low. ATI provides a learning process that guides the decision-making for the next step. Suppose that an internal node q is probed and determined to be dirty. A decision regarding whether to do jumping in q's sub-tree will be made based upon the inspection history. There are two factors contributing to this decision:

- The dirty meter ratio R: obviously, a higher R implies a higher probability of jumping.
- The similarity degree of the inspected nodes that are in the same level with q; this factor is denoted by S_l, which measures the similarity degree in level l. To compute S_l, we assign each node with a dirty meter ratio property called subR, which is the dirty meter ratio of the node's subtree. S_l is given as the standard deviation of subRs of nodes in the same level. Thus, the smaller S_l is, the more similar the nodes are.

Both R and S_l are updated during the inspection process. Based upon the two factors and the previous analysis, the decision process is described as follows:

- when q is a leaf or the parent of a leaf, jumping is unnecessary since it is meaningless;
- if $R < 0.13$, jumping is not applied to q; the value 0.13 is from our analysis result in lemma 5.2.
- if $R \geq 0.13$, jumping is applied. The jumping range D is determined by both R and S_l. we define $D = \left\lfloor \lceil \log_2 n \rceil - l - \frac{S_l}{R} \cdot \left(\lceil \log_2 n \rceil - l \right) \right\rfloor$, which can ensure that $1 \leq D \leq h - l - 1 = \lceil \log_2 n \rceil - l$. Therefore, if S_l is 0, the jumping will take the inspection directly to the leaves. If $S_l / R \geq 1$, there is no need to jump since it indicates that the history does not offer much useful information.

Starting from the root, ATI does a traversal-and-probe with a Depth-First-Search (DFS) algorithm. Specifically, ATI employs the in-order searching order. The reason that one would use DFS is that dirty meters will be determined in the early stage of the entire inspection process; therefore, the information collected during inspection will be more accurate and useful. Once the traversal reaches a leaf node, we need to do probing first, and then update the statistical information of the parent node and the ancestors. The ATI algorithm is described in Table 5.5.

Table 5.5 Adaptive Tree Inspection

Algorithm: Adaptive Tree Inspection (ATI)
Input: the starting *node*, an empty set *Q*
Output: the malicious meter set *Q*
Globals: numMeter := 0, numDirtyMeter := 0, $h = \lceil \log_2 n \rceil + 1$, N[1:h-1]=0, $S[1:h-2] = +\infty$, $Q := \varnothing$, R := 0, *node* := root, D := 0, subR for all nodes are set to 0.
// N_l denotes the number of nodes that are inspected in level l
ATI(*node*, *Q*):

```
    if(node == null)
        return Q;
    if(probe(node) == "dirty")
        if(node is leaf)
            Q := Q ∪ {node}
            numMeter++;
            numDirtyMeter++;
            R = numDirtyMeter / numMeter;
            node.subR = 1;
            //update
            updateAncestor(node);
        else // internal node
            N_node.level++;
            D = decide(R, node.level, S_node.level);
            destNode := node;
            // jumping, if needed
            for(i=0; i <= D; ++i)
                destNode := node.lchild;
                N_destNode.level += 2^(i+1);
            while (destNode is decendent of node)
                ATI(destNode, Q); // recursive call
                destNode := destNode.right;
    else    // node is clean
        if(node is leaf)
            numMeter++;
            R = numDirtyMeter / numMeter;
            node.subR = 0;
        else
            // node is a clean, internal node
            numMeter += 2^(h-node.level)
            R = numDirtyMeter / numMeter;
            updateDecendent(node);
            updateAncestor(node);
// end ATI

updateAncestor(tmp):
    while (tmp is a right child && tmp is not root)
        tmp:= tmp.parent;
        tmp.subR = ½ * (tmp.lchild.subR + tmp.rchild.subR)
```

```
        if (tmp.level <= h - 2)
            S_tmp.level = sqrt( 1/(N_tmp.level - 1) · Σ_{i=1}^{N_tmp.level} (i.subRatio − R)^2 )
// end updateAncestor

updateDecendent(tmp):
    k = 0;
    while(tmp.level <= h-2)
        N_tmp.level += 2^k;
        S_tmp.level = sqrt( 1/(N_tmp.level - 1) · Σ_{i=1}^{N_tmp.level} (i.subRatio − R)^2 )
        tmp := tmp.lchild;
        k++;
//end updateDecendent
```

An example of ATI is given in Fig. 5.8. With a permutation of [0100011010010000], we can track the process in Table 5.6. It is shown that when inspection reaches nodes 5, 3, and 6, decision about jumping needs to be made. Based on the decision-making process, jumping is not applied to both node 5 and 3 because the history information is not sufficient. This explains why we set the initial values of S_l as positive infinity. When node 6 is reached, jumping is applied. In this example, ATI merely saves 2 steps compared to BI_V1. However, when the scale gets large, ATI will be more advantageous.

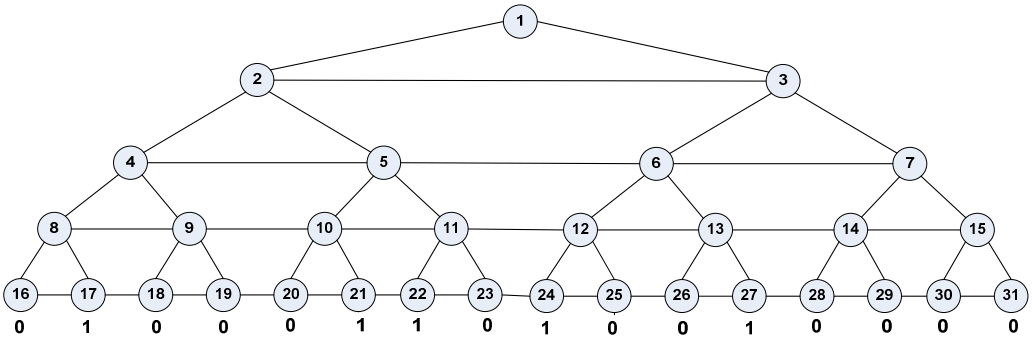


Fig. 5.8 An example of ATI

Table 5.6 ATI tracking example

Probe order	R	subR	S_l $(1 \le l \le 3)$	Decision
1 – 2 – 4 – 8	R=0	--	$S[1:3]=+\infty$	--
16 – 17	R=0.5	8_{subR}=0.5	--	--
9	R=0.25	9_{subR} =0 4_{subR} =0.25	--	--
5	--	--	--	D < 0 No jump
10 – 20 – 21	R=0.33	10_{subR} = 0.5	--	--
11 – 22 – 23	R=0.375	11_{subR} =0.5 5_{subR} =0.5 2_{subR} =0.375	S_3 = 0.177	--
3	--	--	--	D < 0 No jump
6	--	--	--	D = $\lceil 0.54 \rceil$ = 1; Do jumping
24 – 25	R=0.4	12_{subR} =0.5	--	--
26 – 27	R=0.42	13_{subR} =0.5 6_{subR} =0.5	S_3=0.144	--
7	R=0.31	7_{subR} =0 3_{subR} =0.25 1_{subR} =0.31	S_3=0.24 S_2=0.09	--

5.5 Dynamic Inspection

When set Q is dynamic, things become more complicated. For example, a previously checked meter may become malicious and interfere with the inspection. We introduce another assumption for the dynamic inspection: "the detected malicious meter will be removed from the user set immediately so that it will not affect future inspection." The general case of dynamic inspection can be described as follows: there are n end meters in a building; let m denote the eventual size of malicious meter set Q, and the goal is to identify the eventual malicious meter set Q with N_I $(N_I \ge 1)$ inspectors.

5.5.1 Scanning Approach

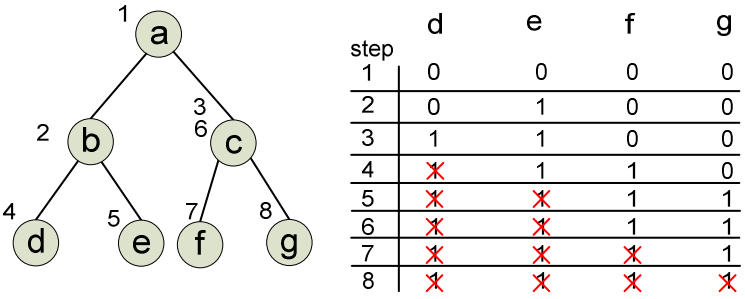
The scanning approach can also be applied to dynamic inspection. The algorithm of the scanning approach for dynamic inspection is given in Table 5.7. The inspection process is divided into rounds. In each round, all meters will be checked one by one. By the end of each

round, we query the head inspector to see if any anomaly exists. If it does, it means that new malicious meters have appeared, and the scanning starts all over again. The inspection terminates when the head inspector reports a normal status after a round of inspection. For dynamic inspection, it takes two or more rounds to finish. Whereas for static inspection, it takes only one round because there will be no malicious meters emerging during inspection.

Table 5.7 Scanning algorithm for dynamic inspection (D-Scanning)

Algorithm: D-Scanning
Input: user set U with size k, inspector set I
Output: malicious meter set Q
Initial: $Q \coloneqq \varnothing$

```
DI_Scan(U):
    if (U is empty)
        return Q;
    while (there is unchecked meter in U)
        for each inspector in I
            if (there is unchecked meter in U)
                take an unchecked meter m_i
                if (probe(m_i) == 'dirty')
                    U := U − {m_i}
                    Q := Q ∪ {m_i}
                Set m_i as checked;
    if (monitor(U) = 'clean')
        return Q;
    else // new malicious meter emerged
        set all meters in U as unchecked
        return DI_Scan(U) ∪ Q                    // start a new round
```



step	d	e	f	g
1	0	0	0	0
2	0	1	0	0
3	1	1	0	0
4	~~1~~	1	1	0
5	~~1~~	~~1~~	1	1
6	~~1~~	~~1~~	1	1
7	~~1~~	~~1~~	~~1~~	1
8	~~1~~	~~1~~	~~1~~	~~1~~

Fig. 5.9 An example of dynamic binary-tree-based inspection with one inspector

5.5.2 Tree-based Dynamic Inspection

The tree-based inspection can be applied to identify the dynamic malicious meter set as well. In this section, we study the dynamic inspection based upon a binary tree (we refer to this scheme as DBI). DBI employs and extends the idea of the static binary-tree-based inspection. The difference between the two is that when DBI finishes a round of inspection, it needs to re-probe the meters that are so far clean until there are no newly emerged bad meters in the current round. Fig. 5.9 shows an example of the dynamic inspection. It can be seen that the dirty meter set Q has been growing from $\varnothing$ to $\{d, e, f, g\}$ during inspection. As a result, some nodes, which in this case is node c, need to be probed multiple times. In this example, the inspection lasts for two rounds. In the 1st round, node a, b, c, d, and e are probed. While c is being checked, f and g are good nodes. After that, f and g become malicious. Therefore, only d and e are identified in the 1^{st} round. After removing d and e from the user set, the 2^{nd} round inspection can start based upon a updated binary tree, which is rooted on node c (because d and e are removed from the tree). When there are multiple inspectors (i.e., $N_I > 1$), the workload can be evenly distributed to each inspector.

The DBI algorithm is described in Table 5.8. It can be seen that once a malicious meter is detected, it will be removed from the tree, which will be re-built after each round. In the algorithm, we employ the basic version of binary inspection (i.e., BI_v1), which can be easily replaced with better strategies.

5.5.3 Algorithm Analysis

In dynamic inspection, m is the eventual size of the malicious meter set Q. We let x_i denote the number of dirty meters detected in the ith round. Let r denote the total number of rounds that the inspection needs to take. Based on our assumption, the dirty meters will be

immediately removed from *U* once detected, and these meters will not be checked in future rounds. We have defined the following term for better illustration.

Table 5.8 Dynamic Binary Inspection (DBI) with $N_I > 1$

```
Algorithm: DBI with N_I > 1
Input: binary tree root, inspector set I
Output: malicious meter set Q
Initial: node := root, Q := ∅, BFSQueue := ∅
DBI(node):
    BFSQueue.enqueue(node);
    while(BFSQueue is not empty)  do
        for each inspector in I
            if (BFSQueue is not empty)
                tmpNode := BFSQueue.dequeue();
                if (probe(tmpNode) = 'dirty')
                    if (tmpNode is leaf)
                    Q := Q ∪ {tmp} ;
                    remove leaf tmpNode from the tree;
                else
                    BFSQueue.enqueue(tmpNode.lchild);
                    BFSQueue.enqueue(tmpNode.rchild);
        if (BFSQueue is empty)
            if (probe(root) = 'clean')
                break; // inspection ends
            else
                // new bad meters emerged
                rebuild the tree with remaining leaves
                update root node
                BFSQueue.enqueue(root); // start over
    return Q;
```

Definition 5.1. *Covered area*: At a particular moment of a specific round, the meters that have been inspected form a set called *covered area*; and the meters that will be checked form a set called *uncovered area*.

Obviously, in the beginning of each round, no meter is in the covered area, and by the end of each round, all of the meters are in the covered area. Once a round is finished, all of the meters belong to the uncovered area again. For dynamic inspection, when there no dirty meter emerging in the covered area after one round, the inspection can be terminated.

1)Scanning approach analysis

Let x_i denote the number of dirty meters detected after round *i* is finished. Table 5.9 is used to track the inspection progress of certain variables. We can then compute the total number of steps (i.e., N_S) by summing up the second column of Table 5.9. We have

$$N_S = r \cdot n - \sum_{l=1}^{r-1} \left(x_l \cdot (r-l) \right) \tag{5.1}$$

Apparently, $m = \sum_{i=1}^{r} x_i < n$, and $x_i > 0$.

Table 5.9 Tracking Inspection progress with D-Scanning

Round	# steps in the ith round	x_i
1	n	x_1
2	$n - x_1$	x_2
…	…	…
r	$n-(x_1+x_2+...+x_{r-1})$	x_r

Lemma 5.5: *given that the size of eventual dirty meter set is m, we have the following bounds of* N_S *for the scanning approach with* $N_I \geq 1$

- *The upper bound of scanning approach for dynamic inspection is* $|N_S|_{\max} = \left\lceil \frac{1}{N_I} \cdot \left(m \cdot n - \frac{1}{2} \cdot (m^2 - m) \right) \right\rceil$, *and this bound can be achieved if and only if* $x_i = 1$ *for all i = 1,2,…, r, (i.e., there is exactly one dirty meter emerging in the covered area in each round).*

- *The lower bound of scanning approach for dynamic inspection is* $|N_S|_{\min} = \left\lceil \frac{1}{N_I} \cdot \left(2 \cdot n - (m-1)\right) \right\rceil$, *and this bound can be achieved if and only if there are (m-1) dirty meters detected in the 1st round, and there is one dirty meter emerging in the covered area in the first round.*

Proof: We will first consider the case when $N_I = 1$. According to equation (5.1) and Table 5.9, the total number of steps is the sum of all elements in a sorted set (in descending order) $S = \{n,\ n - x_1,\ n - (x_1 + x_2),\ \ldots,\ n - (x_1 + x_2 + \ldots + x_{r-1})\}$. If $x_i = 1$ for all i = 1,2, …, r, the size of set S is maximal, and S becomes S_0 = {*n, n-1, n-2, …, n-(m-1)*}. The ith element in S_0 is *n-(i-1)*. Now we will show that no matter how x_i (which is a positive integer) changes its value, the new set *S'* will be a subset of S_0, i.e., $S' \subset S_0$, which means the summation of all elements in S_0 is maximal. We assume that if any particular element $x_i \in S_0$ is increased by Δx, then all the following elements will then be increased by Δx. However, the smallest element will not be less than *n-(m-1)*. Therefore, *S'* is equivalent to S_0 with Δx elements removed. The larger Δx gets, the smaller the size of *S'* will be. For example, let $\Delta x = 5$, and $x_1 = x_1 + \Delta x = 6$, then *S'* becomes {*n, n-6, n-7, …, n-(m-1)*}; in this case, *S'* is obtained by removing 5 elements from S_0. Since the fact that $S' \subset S_0$ always holds, we can show the conditions under which the upper and lower bounds can be achieved: 1) the upper bound of N_S is the sum of all elements in S_0 because any change on x_i will remove elements from S_0, and then decrease N_S. Therefore we have $|N_S|_{\max} = \sum_{t=0}^{m-1} (n - t) = m \cdot n - \frac{1}{2} \cdot (m^2 - m)$; 2) the lower bound of N_S can be achieved if we

remove elements from S_0 as many as we can. Apparently, when $x_1 = m-1$, $x_2 = 1$, we have $S' = \{n, n-(m-1)\}$, which is the smallest S' we can get. Therefore, $|N_S|_{\min} = 2 \cdot n - (m-1)$.

Since in above we have shown that when $N_I = 1$, $|N_S|_{\max} = m \cdot n - \frac{1}{2} \cdot (m^2 - m)$, and $|N_S|_{\min} = 2 \cdot n - (m-1)$. we can infer that when $N_I > 1$, the bounds would be $\lceil |N_S|_{\max} / N_I \rceil$ and $\lceil |N_S|_{\min} / N_I \rceil$, respectively, because the workload will be shared by N_I inspectors. □

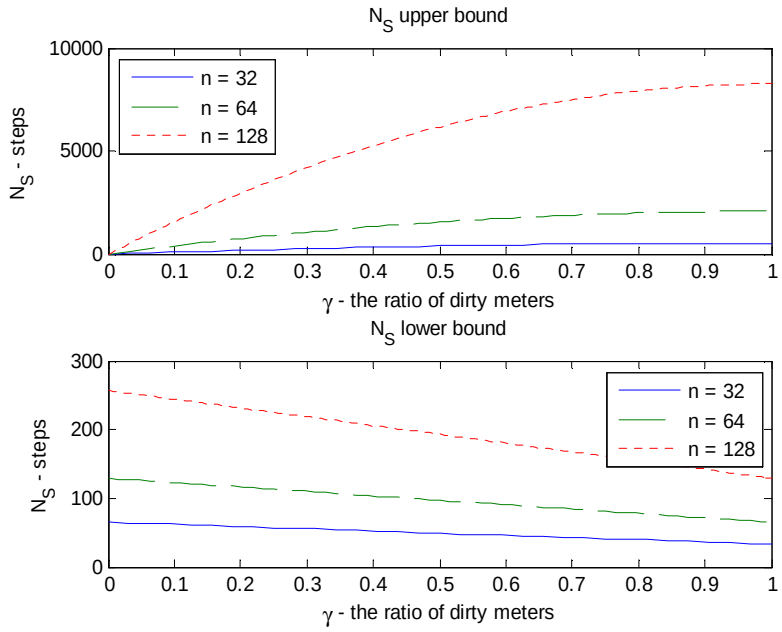


Fig. 5.10 The upper bound (top) and lower bound (bottom) of N_S for D-Scanning

The numeric results of the bounds of N_S of D-Scanning are shown in Fig. 5.10.

5.6 Evaluation

5.6.1 Static Inspection

The numeric performance bounds of N_S will only be achieved in specific conditions. In the real world, the ratio of dirty meters and their distribution pattern remain unknown before

inspection, and the performance bounds may not be achieved in regular cases. In this section, we evaluate the performance when the dirty meters are randomly deployed in static inspection. We set $n = 512$ and $N_I = 1$. Each piece of data is based upon the average value of 30 repeats. The main result is described in Fig. 5.11 from which we can see that BI_v2 is better than BI_v1. The binary inspection approach (i.e., BI_v1 and BI_v2) is better than scanning when the dirty meter ratio is low; however, when the ratio increases, scanning still remains constant while the binary inspection is getting worse. In addition, the adaptive tree approach has a clear advantage over the prior two approaches since it adjusts the inspection strategy when traversing the tree based upon the heuristic information.

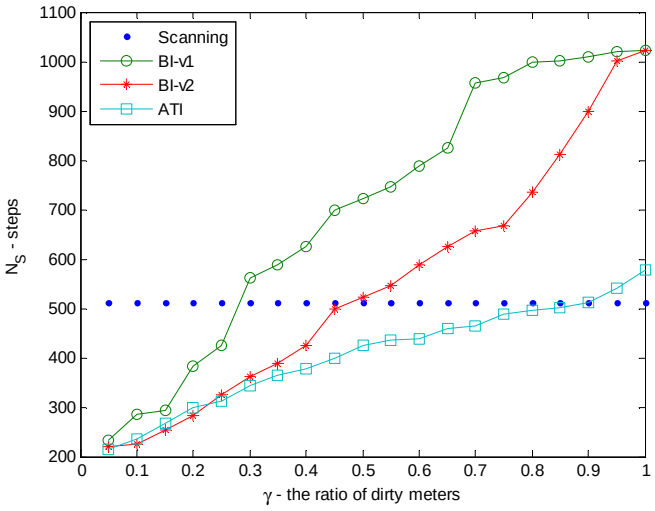


Fig. 5.11 Evaluation results when *n* = 512

5.6.2 Dynamic Inspection

We have determined the bounds of N_S when different approaches are employed. However, the dynamic inspection is featured with high uncertainty. That is why it should be studied with further evaluation. Given that n is the number of user meters, m is the size of the

eventual dirty meter set, and *r* is the number of rounds needed to finish the dynamic inspection. The evaluation is designed to observe the average performance by adjusting the three parameters.

In Fig. 5.12, we let *n* = 128, and the goal is to test how *m*, the dirty meter ratio (i.e., γ), affects the performance (i.e., N_S. In this test, *m* is presented by dirty meter ratio) (i.e, $\gamma = m/n$) ranging from 0.1 to 1, and the dirty meters are randomly deployed in the *n* meters. For each time of test, we let *m* fixed and test the performance by increasing *r* from 2 to *m*. After averaging the results of 20 tests, we find out that when γ increases, the overall trend is rising no matter which approach is chosen. Comparing the different approaches, we observe that DBI_v2 is better than DBI_v1, but their curve is close, because the minor optimization in DBI_v2 happens with a probability. Also, both DBI v1 and v2 are much better than the scanning approach due to the multi-round property in the dynamic case. Since the *m* dirty meters are split into multiple rounds, for each round, the number of bad meters may be small. Adaptive-tree inspection presents a better performance than the previous three. We conclude that in the real world situation, scanning has already lost its advantage.

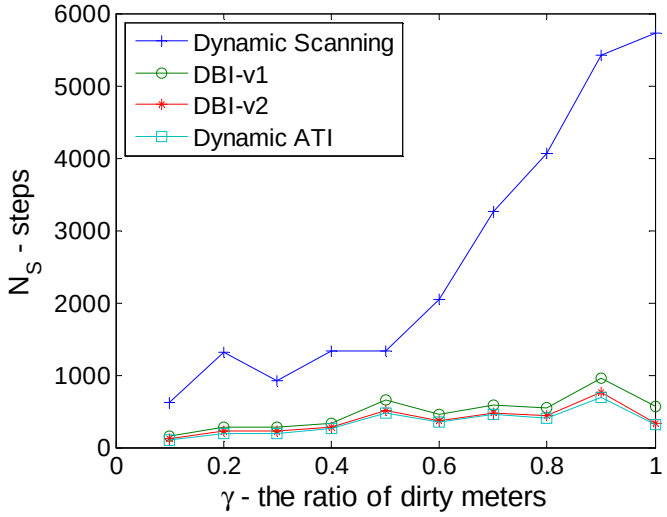


Fig. 5.12 The average performance of DBI_v1,DBI_v2, D-scanning, and D-ATI when *n* = 128

To determine how rounds affect the performance, we have designed another test, in which both *n* and *m* are fixed. We let *n* = 128, and *m* = 24, and we examine the performance when *r* is increased from 2 to m. The result is shown in Fig. 5.13, in which the curves are all climbing. For a specific *r*, the tree-based inspection schemes are far better than D-Scanning. Especially when *r* is getting larger, the performance gap between the tree-based inspection and scanning becomes wider. The reason is that since both *n* and *m* are constant, when *r* gets larger, the number of dirty meters for each round will decrease, and D-Scanning is less efficient. Therefore, D-Scanning only outperforms the tree-based inspection when *m* is sufficiently large.

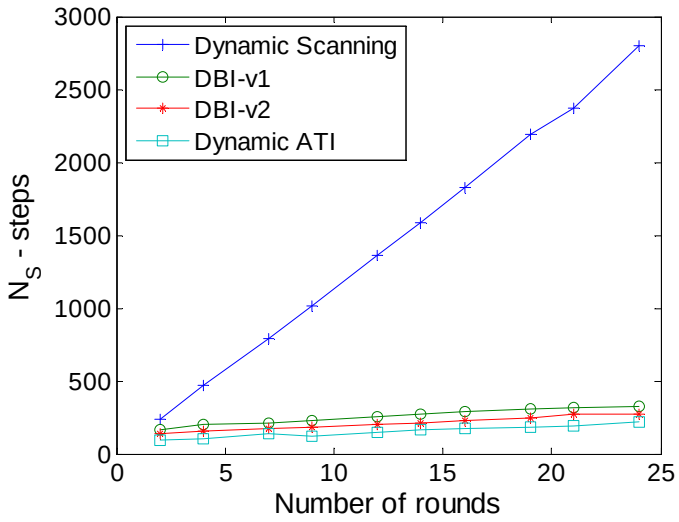


Fig. 5.13 The number of rounds Vs. the number of steps when *n* = 128 and *m* = 24

5.7 Conclusion

In this chapter, we have discussed the MMI problem in the neighborhood area smart grid. We have formulized the MMI problem and analyzed the solution algorithms that are based on an inspection tree. We have analyzed and compared the performance of the proposed algorithms. It turns out that the adaptive tree inspection scheme is superior to other algorithms, because it

utilizes the heuristic information and can adjust the tree-walking in real time. Based on our study, we find that static inspection is useful in the situation that once a round of inspection completes, the system is back to normal. Dynamic inspection is applicable when anomaly is detected right after a round of inspection completes. Each time there will be only one algorithm running. The smart grid operator is responsible for determining which one to use according to the situation.

CHAPTER 6

CONCLUSION

The research presented in this dissertation addresses how to design and analyze an accountable environment for distributed and networked systems. Throughout the study, we focus on incorporating accountability into the design and implementation of modern computing systems with affordable expense. Specifically, we have studied accountability in traditional distributed systems, cloud computing, and the Smart Grid, each of which poses particular research challenges.

The dissertation explores a fundamental problem that involves the assessment of the degree of system accountability. To address the problem, we propose P-Accountability as a quantitative model. P-Accountability consists of a flat model and a hierarchical model, which can be chosen to use depending on the complexity of the system. We demonstrate that P-Accountability is applicable to PeerReview and its extended version, i.e., Traceable PeerReview. Our simulation results show that P-Accountability can effectively reflect the degree of accountability for systems supported by PeerReview and TPR.

We also propose Accountable MapReduce in the cloud environment. The purpose of Accountable MapReduce is to hold each cloud working machine responsible for its behavior. We set up a group of auditors to perform an Accountability-Test (A-test) that checks all working machines and detect malicious nodes in real time. To optimize the resource utilization, we formulize the Optimal Worker and Auditor Assignment (OWAA) problem, which aims to find the optimal number of workers and auditors in order to minimize the total processing time. Our

evaluation results show that the A-test can be practically and effectively applied to existing cloud-based MapReduce systems.

Finally, we study the accountability issue in the Smart Grid. We first present a mutual inspection strategy to enable non-repudiation on meter reading for the Smart Grid in NAN. The goal of our scheme is to discover problematic meters that report inaccurate readings. We then define and explore the Malicious Meter Inspection (MMI) problem. We have proposed, analyzed, and evaluated a suite of algorithms. In addition, we have compared the proposed algorithms in terms of a metric, i.e., the number of steps the inspection takes. We discover that the adaptive-tree-based algorithm is superior to other algorithms because it saves most steps in an average case.

Accountability has become a first class design principle for modern computing systems. In our future work, we will explore the accountability issues for the state-of-the-art computing environments, for instance, Internet of things, data center network, social network, etc.

REFERENCES

[1] A. Haeberlen, P. Kouznetsov, and P. Druschel, "*PeerReview: Practical accountability for distributed systems*," Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, ACM New York, NY, USA, pp. 175-188, 2007.

[2] Department of Defense. "*Trusted Computer System Evaluation Criteria*," Technical Report 5200.28-STD, 1985.

[3] A.R. Yumerefendi and J.S. Chase, "*The role of accountability in dependable distributed systems*," Proceedings of the First conference on Hot topics in system dependability (HotDep'05). USENIX Association, Berkeley, CA, USA, 2005.

[4] A.R. Yumerefendi and J.S. Chase, "*Trust but verify: accountability for network services*," Proceedings of the 11th workshop on ACM SIGOPS European workshop, Leuven, Belgium: ACM, p. 37, 2004.

[5] Y. Xiao, "*Flow-Net Methodology for Accountability in Wireless Networks*," *IEEE Network*, Vol. 23, No. 5, pp. 30-37, Sept./Oct. 2009.

[6] D. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker, "*Holding the Internet accountable*," ACM HotNets-VI, 2007.

[7] D.G. Andersen, N. Feamster, T. Koponen, D. Moon, and S. Shenker, "*Accountable internet protocol (AIP)*," Proceedings of the ACM SIGCOMM 2008 conference on Data communication, ACM New York, NY, USA, pp. 339-350, 2008.

[8] J. Mirkovic and P. Reiher, "*Building accountability into the future Internet*," 4th Workshop on Secure Network Protocols (NPSec 2008), pp. 45-51, 2008.

[9] M. Backes, P. Druschel, A. Haeberlen, and D. Unruh, "*CSAR: A practical and provable technique to make randomized systems accountable*," Proceedings of the 16th Annual Network & Distributed System Security Symposium (NDSS '09), San Diego, CA, February 2009.

[10] E. Keller, R. Lee, and J. Rexford, "*Accountability in hosted virtual networks*," Proceedings of the ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures (VISA), August 2009.

[11] W. Liu, S. Aggarwal, and Z. Duan, "*Incorporating accountability into internet email*," Proceedings of the 2009 ACM symposium on Applied Computing, pp. 875–882, 2009.

[12] K. Argyraki, P. Maniatis, O. Irzak, S. Ashish, S. Shenker, and L. EPFL, "*Loss and delay accountability for the Internet*," IEEE International Conference on Network Protocols (ICNP 2007), pp. 194–205, 2007.

[13] A.R. Yumerefendi and J.S. Chase, "*Strong accountability for network storage*," ACM Transactions on Storage, Vol. 3, 2007.

[14] B. Briscoe, A. Jacquet, T. Moncaster, and A. Smith, "*Re-ECN: Adding Accountability for Causing Congestion to TCP/IP*," IETF Internet-Draft, Mar 2009.

[15] B. Briscoe, A. Jacquet, T. Moncaster, and A. Smith, "*Re-ECN: The Motivation for Adding Accountability for Causing Congestion to TCP/IP*," IETF Internet-Draft, Mar 2009.

[16] L. Zeng, H. Chen, and Y. Xiao, "*Accountable Administration and Implementation in Operating Systems*," Proceedings of IEEE GLOBECOM 2011.

[17] B.B. Madan, K. Goeva-Popstojanova, K. Vaidyanathan, and K.S. Trivedi, "*A method for modeling and quantifying the security attributes of intrusion tolerant systems*," Performance Evaluation, Vol. 56, pp. 167–186, 2004.

[18] R. Breu, F. Innerhofer-Oberperfler, and A. Yautsiukhin, "*Quantitative assessment of enterprise security system*," Third International Conference on Availability, Reliability and Security (ARES 08), pp. 921–928, 2008.

[19] K. Sallhammar, B. Helvik, and S. Knapskog, "*A Game-Theoretic Approach to Stochastic Security and Dependability Evaluation*," Dependable, Autonomic and Secure Computing, 2nd IEEE International Symposium on, pp. 61-68, 2006.

[20] G. Bella and L. C. Paulson, "*Accountability Protocols: Formalized and Verified*," ACM Trans. on Information and System Security, Vol. 9, No. 2, pp. 138–161, 2006.

[21] G. Bella, "*Inductive Verification of Cryptographic Protocols*," Ph.D. thesis, Research Report 493, Computer Laboratory, University of Cambridge, 2000.

[22] R. Jagadeesan, A. Jeffrey, C. Pitcher, and J. Riely, "*Towards a theory of accountability and audit*," Proceedings of ESORICS'09, volume 5789 of LNCS, pp. 152–167, Springer, 2009.

[23] R. Küsters, T. Truderung, and A. Vogt, "*Accountability: definition and relationship to verifiability*," Proceedings of the 17th ACM conference on Computer and communications security, New York, NY, USA, pp. 526–535, 2010.

[24] J. Feigenbaum, A. D. Jaggard, and R. N. Wright, "*Towards a formal model of accountability*," Proceedings of the 2011 workshop on New security paradigms workshop, New York, NY, USA, pp. 45–56, 2011.

[25] Z. Xiao and Y. Xiao, "*PeerReview Analysis and Re-evaluation for Accountability in Distributed Systems or Networks*", Proceedings of The 4th International Conference on Information Security and Assurance (ISA2010), 2010

[26] PeerReview software, available at: http://peerreview.mpi-sws.org/

[27] V. Paxson, "*End-to-end Internet packet dynamics*," SIGCOMM Comput. Commun. Rev., Vol. 27, pp. 139-152, 1997.

[28] Z. Xiao, Y. Xiao, and J. Wu, "*A Quantitative Study of Accountability in Wireless Multi-hop Networks*," Proceedings of the 39th International Conference on Parallel Processing (ICPP), pp. 198–207, 2010.

[29] J. Liu, Y. Xiao, and G. Gao, "*Achieving Accountability in Smart Grids*," IEEE systems journal, accepted.

[30] R. Santamarta, "*HERE BE BACKDOORS: A Journey Into The Secrets Of Industrial Firmware*," Black Hat USA 2012, July, Las Vegas.

[31] Y. Xiao, K. Meng, and D. Takahashi, "*Accountability using Flow-net: Design, Implementation, and Performance Evaluation*," (Wiley Journal of) Security and Communication Networks, Vol.5, NO. 1, pp. 29–49, Jan. 2012, DOI: 10.1002/sec.348.

[32] J. Dean and S.Ghemawat, "*Mapreduce: simplified data processing on large clusters*," Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation (OSDI'04), Berkeley, CA, USA, 2008

[33] Apache™ Hadoop, available at: http://hadoop.apache.org/

[34] Z. Xiao and Y. Xiao, "*P-Accountable Networked Systems*," INFOCOM IEEE Conference on Computer Communications Workshops , 2010.

[35] W. Wei, J. Du, T. Yu, and X. Gu, "*SecureMR: A Service Integrity Assurance Framework for MapReduce*," Proceedings of the 2009 Annual Computer Security Applications Conference, pp. 73–82, 2009.

[36] I. Roy, S. Setty, A. Kilzer, V. Shmatikov, and E. Witchel, "*Airavat: Security and privacy for MapReduce*," Proceedings of the USENIX Symposium on Networked Systems Design and Implementation, pp. 297–312, 2010.

[37] J. Schlesinger, "*Cloud Security in Map/Reduce*," 2009, available at: http://www.defcon.org/images/ defcon-17/dc-17-presentations/defcon-17-jason_schlesinger-cloud_security.pdf

[38] C.T. Chu, S.K. Kim, Y.A. Lin, Y.Y. Yu, G. Bradski, A.Y. Ng, and K. Olukotun, "*Map-reduce for machine learning on multi-core*," Advances in Neural Information Processing Systems, pp. 281 - 288, 2007.

[39] A. Wieder, P. Bhatotia, A. Post, and R. Rodrigues, "*Brief announcement: modeling MapReduce for optimal execution in the cloud*," Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing, pp. 408–409, 2010.

[40] D. Gottfrid, "*Self-service, Prorated Super Computing Fun*!" The New York Times, available at: http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-un /?scp =1 & sq= self%20 service%20 prorated&st= cse.

[41] B. He, W. Fang, Q. Luo, N.K. Govindaraju, and T. Wang, "*Mars: a MapReduce framework on graphics processors*," Proceedings of the 17th international conference on Parallel architectures and compilation techniques, pp. 260–269, 2008.

[42] J. Ekanayake, S. Pallickara, and G. Fox, "*MapReduce for data intensive scientific analyses*," IEEE Fourth International Conference on eScience (eScience'08), pp. 277–284, 2008.

[43] S. Papadimitriou and J. Sun, "*Disco: Distributed co-clustering with Map-Reduce: A case study towards petabyte-scale end-to-end mining*," Eighth IEEE International Conference on Data Mining (ICDM'08), pp. 512–521, 2008.

[44] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, and others, "*Above the clouds: A Berkeley view of cloud computing*," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, 2009.

[45] A. Haeberlen, "*A case for the accountable cloud*," ACM SIGOPS Operating Systems Review, Vol. 44, pp. 52–57, 2010.

[46] C. Wang and Y. Zhou, "*A Collaborative Monitoring Mechanism for Making a Multitenant Platform Accountable*," Proceedings of the 2nd USENIX conference on Hot topics in cloud computing (HotCloud'10), USENIX Association, Berkeley, CA, USA, pp. 18-18, 2010.

[47] A. Juels and B. S. Kaliski, "*PORs: Proofs of retrievability for large files*," ACM CCS, pages 584–597, 2007.

[48] H. Farhangi, "*The path of the smart grid*," Power and Energy Magazine, IEEE, Vol. 8, No. 1, pp. 18–28, 2009.

[49] X. I. E. Kai, L. I. U. Yong-qi, Z. H. U. Zhi-zhong, and Y. U. Er-keng, "*The Vision of Future Smart Grid*," Electric Power, Vol. 41, No. 6, pp. 19–22, 2008.

[50] M. Amin and B. F. Wollenberg, "*Toward a smart grid: power delivery for the 21st century*," Power and Energy Magazine, IEEE, Vol. 3, No. 5, pp. 34–41, 2005.

[51] Accenture Inc., "*Achieving high performance with theft analytics - Leveraging smart grid employments to enhance revenue protection*," 2011, available at: http://www.accenture.com/SiteCollectionDocuments/PDF/Accenture-Achieving-High-Performance-with-Theft-Analytics.pdf

[52] P. McDaniel and S. McLaughlin, "*Security and Privacy Challenges in the Smart Grid*," IEEE Security & Privacy, Vol. 7, No. 3, pp. 75-77, 2009.

[53] S. McLaughlin, D. Podkuiko, and P. McDaniel, "*Energy Theft in the Advanced Metering Infrastructure*," Critical Information Infrastructures Security Lecture Notes in Computer Science, 2010, Volume 6027/2010, 176-187, DOI: 10.1007/978-3-642- 14379-3_15

[54] M. Madrazo, "*Today's Energy Theft Detection Models Help Protect Revenues While Enhancing Neighborhood Safety*," Pipeline & Gas Journal, Vol. 237, No. 7, July 2010, available at: http://www.pipe-lineandgasjournal.com/today's-energy-theft-detection-models-help-protect-revenues-while-enhancing-neighborhood-safety

[55] S.S.S.R. Depuru, L. Wang, and V. Devabhaktuni, "*Support vector machine based data classification for detection of electricity theft*," Power Systems Conference and Exposition (PSCE), 2011.

[56] J. Liu, Y. Xiao, and J. Gao, "*Accountability in Smart Grids*," IEEE Consumer Communications and Networking Conference 2011 (IEEE CCNC 2011), Smart Grids Special Session.

[57] Z. Xiao, Y. Xiao, and D. Du, "*Building Accountable Smart Grids in Neighborhood Area Networks*," Proceedings of IEEE GLOBECOM 2011.

[58] "*Cisco Smart Grid Security Solutions Brief*," 2009 Cisco Systems, Inc. available at: http://www.cisco.com/web/strategy/docs/energy/CiscoSmartGridSecurity_solutions_brief_c22-556936.pdf

[59] Y. Xiao and J. Rosdahl, "*Throughput and delay limits of IEEE 802.11*," Communications Letters, IEEE, Vol. 6, pp. 355–357, 2002.

[60] R. E. Brown, "*Impact of Smart Grid on distribution system design*," Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE, pp. 1–4, 2008.

[61] C. Shuyong, S. Shufang, L. I. Lanxin, and S. Jie, "*Survey on smart grid technology*," Power System Technology, Vol. 33, No. 8, pp. 1–7, 2009.

[62] "*Reducing revenue leakage*," Electric Light and Power Magazine, available at: http://uaelp.pennnet.com/, 2009.

[63] D. Du and F.K. Hwang, "*Combinatorial Group Testing and its Applications*," World Scientific, Singapore, 1993.

[64] R. Dorfman, "*The detection of defective members of large populations*," Ann. Math. Statist. Vol. 14, pp. 436–440, 1943.

[65] C. J. Bandim, J. E. R. Alves, A. V. Pinto, F. C. Souza, M. R. . Loureiro, C. A. Magalhaes, and F. Galvez-Durand, "*Identification of energy theft and tampered meters using a central observer meter: a mathematical approach*," Transmission and Distribution Conference and Exposition (2003 IEEE PES), Vol. 1, pp. 163- 168, 2003.

[66] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, "*Introduction to Algorithms*," second edition, Chapter 4, Section 4.2, 2001.

[67] Y. Xiao, "*Editorial*," International Journal of Security and Networks, special issue on Security and Privacy in Smart Grid, Vol. 6, No.1, pp. 1 - 1, 2011.

[68] D. Kundur, X. Feng, S. Mashayekh, S. Liu, T. Zourntos, K.L. Butler-Purry, "*Towards modeling the impact of cyber attacks on a smart grid*," International Journal of Security and Networks, Vol. 6, No.1, pp. 2 - 13, 2011.

[69] G. Kalogridis, S.Z. Denic, T. Lewis, R. Cepeda, "*Privacy protection system and metrics for hiding electrical events*," International Journal of Security and Networks, Vol. 6, No.1, pp. 14 - 27, 2011.

[70] F. Li, B. Luo, P. Liu, "*Secure and privacy-preserving information aggregation for smart grids*," International Journal of Security and Networks, Vol. 6, No.1, pp. 28 - 39 , 2011.

[71] J. Zhang and C. A. Gunter, "*Application-aware secure multicast for power grid communications*," International Journal of Security and Networks, Vol. 6, No.1, pp. 40 - 52, 2011.

[72] J. Gao, J. Liu, B. Rajan, R. Nori, B. Fu, Y. Xiao, W. Liang, and C. L. P. Chen, "*SCADA Communication and Security Issues*," (Wiley Journal of) Security and Communication Networks, accepted.

[73] Z. Xiao, Y. Xiao, and D. Du, "*Non-repudiation in Neighborhood Area Networks for Smart Grid*," IEEE Communications Magazine, Vol. 51, No. 1, pp. 18-26, Jan. 2013.

[74] J. Liu, Y. Xiao, S. Li, W. Liang, C. L. P. Chen, "*Cyber Security and Privacy Issues in Smart Grids*," IEEE Communications Surveys & Tutorials, Vol. 14, NO. 4, pp. 981 - 997, Fourth Quarter 2012. DOI: 10.1109/SURV.2011.122111.00145.

[75] J. Gao, Y. Xiao, J. Liu, W. Liang, and C. L. P. Chen, "*A Survey of Communication/ Networking in Smart Grids*," (Elsevier) Future Generation Computer Systems, Vol. 28, No. 2, pp. 391–404, Feb. 2012.

[76] Utah Emulab, http://www.emulab.net/