Exploring Malicious Meter Inspection in Neighborhood Area Smart Grids

Zhifeng Xiao, Yang Xiao, and David Hung-Chang Du

***Abstract*—In smart grids, smart meters may potentially be attacked or compromised to cause certain security risks. It is challenging to identify malicious meters when there are a large number of users. In this paper, we explore the malicious meter inspection (MMI) problem in neighborhood area smart grids. We propose a suite of inspection algorithms in a progressive manner. First, we present a basic scanning method, which takes linear time to accomplish inspection. The scanning method is efficient when the malicious meter ratio is high. Then, we propose a binary-tree-based inspection algorithm, which performs better than scanning when the malicious meter ratio is low. Finally, we employ an adaptive-tree-based algorithm, which leverages advantages of both the scanning and binary-tree inspections. Our approaches are tailored to fit both static and dynamic situations. The theoretical and experimental results have shown the effectiveness of the adaptive tree approach.**

***Index Terms*—Accountability, advanced metering infrastructure (AMI), attack, malicious meter inspection, security, smart grid, smart meter.**

I. Introduction

SMART grids have received significant attentions in recent advances [1]–[6]. A smart grid delivers electricity from providers to users, and it uses two-way digital communications to control appliances at users' homes; this saves energy, reduces costs, and increases reliability and transparency. However, security in smart grids is also becoming a significant concern. New vulnerabilities are introduced when new hardware and software techniques (e.g., advanced metering system and digital networks) are brought in. One of the serious issues is energy theft or malicious attacking, which has been a chronic problem and still exists in smart grid. For example, hackers who compromise a meter can immediately manipulate the amount of service or fabricate generated energy meter readings [7]. It has been reported that annual losses due to theft of service in the United States alone are estimated as 6 billion dollars [9]. Therefore, it is imperative to develop countermeasures to prevent energy theft in smart grid.

Back in the 20th century, power providers employed meter readers to do a door-to-door meter reading. There are many drawbacks of artificial meter reading, such as 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. In other words, if a meter is tampered and the reading value is less than the actual amount, 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. However, AMI could not ensure non-repudiation of meter reading as well, where non-repudiation refers to a state where one party cannot deny what its activities were. The root problem of non-repudiation lies in the way of collecting the reading values of smart meters. In order to acquire the service amount of each user, the power provider has to rely on the digital communication network for data transmission. Since the reading values are generated in the user end, an attacker or an energy thief still has multiple means to tamper with them. The most common methods [12] 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, etc. The original reading may be altered before it is sent to the provider. Since the smart meter may be the only source for the power provider to acquire the service amount, no matter if the meter reading is accurate or falsified, the power provider has no means to verify the correctness of the meter's reading report.

To solve this issue, we have proposed a mutual inspection scheme [8], [23] which installs a redundant meter in the provider end to inspect each user. This means that for each individual power line, there is one smart meter on each end. The difference between the two reading values should be within a certain range. Otherwise, a dispute will arise, indicating that the smart meter may be under attack.

One limitation of the mutual inspection scheme is its cost. Adding a redundant meter for each user will increase the budget on both hardware and management. The cost issue is magnified in the metropolitan area (e.g., the New York City, Beijing, Tokyo, etc); the most common living style in big cities is an apartment building assembling multiple apartments. A large building could consist of hundreds of apartment cells. According to mutual inspection, the power provider should install a redundant smart meter (referred to as an **inspector**) at the provider's end of the power line for inspection purposes. Thus, the number of inspectors is equal to the number of users. In this paper, we consider the situation in which the number of inspectors is far less than the number of smart meters to be checked.

Manuscript received March 31, 2012; revised September 07, 2012; accepted October 02, 2012. Date of publication December 28, 2012; date of current version February 27, 2013. This work was supported in part by the US National Science Foundation (NSF) under Grants CNS-0737325, CNS-0716211, CCF-0829827, and CNS-1059265.

Z. Xiao and Y. Xiao are with the Department of Computer Science, University of Alabama, Tuscaloosa, AL 35487-0290 USA (e-mail: zxiao1@crimson.ua.edu; yangxiao@ieee.org).

D. H.-C. Du is with the Department of Computer Science, University of Minnesota, Minneapolis, MN 55455 USA (e-mail: du@cs.umn.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSG.2012.2229397

Our main contributions are summarized as follows: Firstly, we propose the malicious meter inspection (MMI) problem in this paper. To our knowledge, our work is the first time in which the MMI problem is formulated. Secondly, we investigate the MMI problem (including static and dynamic cases) in a smart grid of a neighborhood area. The goal of this paper is to reduce inspection cost and maintain high efficiency at the same time. Thirdly, we propose a suite of algorithms to solve the MMI problem in both static and dynamic cases. The algorithms we adopt in this paper 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 numeric and experimental results for the discussed algorithms. It turns out that the adaptive tree approach is a decent choice in general circumstances.

The rest of this paper is structured as follows: Related works are presented in Section II. The malicious meter inspection problem is formularized in Section III. Static and dynamic inspections are discussed in Section IV and Section V, respectively. We evaluate the algorithms in Section VI and conclude this paper in Section VII.

II. Related Work

Energy theft is the main incentive to compromise a smart meter. McLaughlin *et al.* [12] demonstrate that not only is energy theft possible in the Advanced Metering Infrastructure (AMI) system but that the AMI commodity devices can be taken advantage of by adversaries. There are three classes of attacks based on when and where the data for the amount of service is manipulated. They include: 1) while it is recoded, 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 [11]: 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 incorrectness in energy metering. Characteristic analysis, on the other hand, attempts to model the consumption pattern for an account; thus, any anomalies not following the pattern may be indicative of energy theft. In this area, machine learning techniques (e.g., SVMs [10]) can be leveraged to build fundamental patterns and detect anomalies. However, these analytical methods may not be properly used as evidence of energy theft because a deviation from expected normal usage can be caused by reasons other than energy theft. For example, one needs to consider the trend of energy usage in the entire area or other legitimate changes. In this paper, our method differs from the early analytical methods in the way that we target to isolate the compromised meter(s) with undeniable evidence, which can be used as proof of misbehavior.

Bandim *et al.* [15] 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) so that 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 if an attacker changes its strategy, then the method would not work. There are also some other related work [1]–[26].

In this paper, we provide a comprehensive solution that covers multiple attack models with undeniable evidence to identify the malicious meters.

TABLE I
NOTATIONS

Notation	*Description*
U	A set of users.
n	$\lvert U \rvert := n$, the total number of users (smart meters).
Q	A set of malicious users. We also have $Q \subseteq U$.
m	$\lvert Q \rvert := m$, 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 $Perm_U := [v_1 v_2 \dots v_n]$ in which v_i is binary. $v_i = 0$ means user u_i is good, and $v_i = 1$ means it is malicious.
γ	Malicious meter ratio, given by $\gamma = m/n$

III. Malicious Meter Inspection Problem

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

Smart grid model: Consider an apartment building that consists of n users (i.e., each apartment represents a user) that compose a user set U. Each user has one meter installed to measure the service amount. In this paper, 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 one central monitor for each building. If there is any meter reading error, 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 report an abnormal condition to the provider.

Attack model: McLaughlin *et al.* [12] have investigated the attack models of energy theft with an attack tree, which has categorized the approaches to commit energy theft. Based upon their security analysis and case study on an experimental test-bed, it is practically viable to tamper with a smart meter's measured data. In this paper, 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. In this paper, we consider two kinds of attack models, which have covered the ones discussed in [12]. (1) **Independent-meter Attack**: The target is only one smart meter, which could be attacked by external adversaries or be manipulated by malicious users. 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 data so that victims will have excessive power bills. Note that in this attack models, multiple

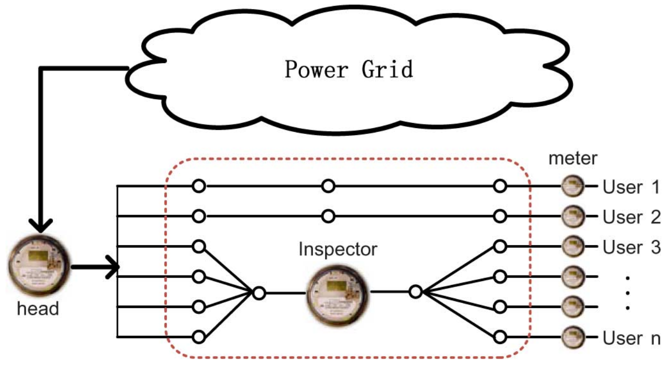


Fig. 1. An inspector box located outside each apartment building.

meters can be attacked independently. (2) **Collaborate-meter Attack**: In this case, victims are multiple meters that are controlled or compromised by some criminal organizations. The purpose of a Collaborate-meter attack is to use the behavior of a meter group to mask the anomaly of an individual meter. For example, to bypass the head inspector, a malicious user that controls multiple meters may decrease his own reading data and increase other meters' reading data. The total demand of these meters does not change, and head inspector is unable to detect any misbehavior. We need to point out that this attack may be specific to our smart grid model.

In this paper, we provide a full solution for the independent-meter attack. The problem of the Collaborate-meter attack will be addressed in our future work.

A. Problem Statement

For inspection purposes, 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. 1 describes a possible look for the inspector box with one inspector.

The job of the head inspector is to detect that a reading anomaly exists 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 paper, we develop a multi-step strategy to identify the compromised meter(s) with a limited number of inspectors.

There are two parameters for 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); In each step, the inspection will last for a while because it is attempting to identify all malicious meters. (2) The number of inspectors (denoted as N_I), excluding the head inspector.

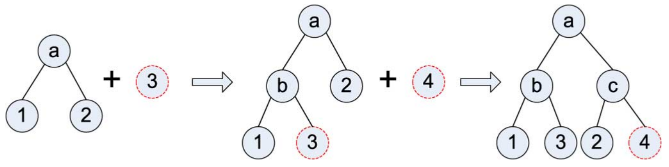


Fig. 2. PullDown operation.

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 the malicious user set, 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 entire inspection process; dynamic inspection considers a changeable Q in the inspection process. In this paper, we investigate **MMI** in both static and dynamic inspection.

B. MMI and Group Testing

To some extent, the **MMI** problem is similar to the group testing problem [14], which was invented by R. Dorfman in World War II in order to facilitate the procedure of identifying syphilis in blood samples from millions of draftees. Du *et al.* have summarized the group testing problem and its variations in [13]. A plenty of prior works have been done regarding the algorithm improvements, problem variations, special case discussions, etc. 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 amount of time. The MMI problem can be regarded as a variation of group testing with dynamic feature. Furthermore, our proposed solutions are different from existing group testing methods.

IV. Static Inspection

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

A. Scanning Approach—A Brute-Force Strategy

When set Q is static, the naïve solution is to scan every meter once. 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. 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 inspector resources. By taking advantage of the

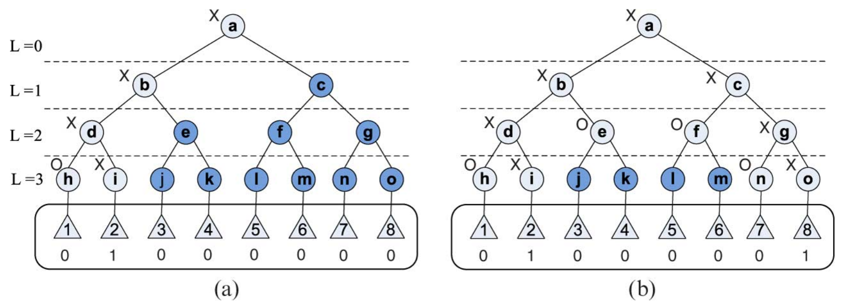


Fig. 3. (a) An example of binary inspection with $\mathrm{Perm_U} = [01000000]$, $\mathrm{m} = 1$, and $N_I = 1$; (b) $\mathrm{Perm_U} = [01000001]$, $m = 2$, and $N_I = 1$.

TABLE II
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';
```

smart grid property, we have designed a tree-based inspection scheme, which employs a tree as a *logic* structure to identify the compromised meters.

B. 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 meters 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. 3(a), if node b is an inspector, it is able to inspect meters 1, 2, 3, and 4, which all belong to the subtrees of node b. An internal node can tell whether there is any anomaly in the subtree, but it does not know exactly which one is bad; that means 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. 3 and other related figures where the inspection tree is applied, symbol 'X' means an anomaly is detected (we call the node **dirty**), and 'O' means the meter is working correctly (we call the node **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. A node's status can be determined by probing (as shown in Table II).

In Table II, we define a probing operation that 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 that is basically determined by the power loss during transmission. Specifically, $\delta(u_i)$ denotes the threshold of user u_i. According to electricity knowledge, the amount of lost power depends on the wire resistance, the transmission voltage, and the power demand of user. Also, 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 threshold will not be discussed since it is not the focus of this paper. If the difference between $R(x)$ and the sum of reported readings from user meters is larger than the sum of the threshold values, then the node is considered '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: In this paper, the tree is built when needed. In other words, when an internal node is probed clean, then its subtree is not built since all users under its inspection are clean; when an internal node is probed dirty, then the node forks two branches for further inspection. Given a user set U, we build a binary tree with the n meters randomly picked as leaf nodes. For any n, we have $2^i \leq n < 2^{i+1}$ $(i \geq 0)$. When $n = 2^i$, the binary tree is complete. Otherwise, it is incomplete. The tree-building process is described as follows:

- If $n = 2^i$, then a complete binary tree is built with height $\log_2 \mathrm{n} + 1$. 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. 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.

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 before inspection initiates. However, it presents the basic idea of the inspection scheme. In Fig. 3(a), root a is detected dirty due to meter 2. The inspection is performed in a top-down pattern.

Based upon Algorithm BI_basic (shown in Table III), we can determine the upper bound and lower bound of N_S. Since the height of the binary tree is $\lceil \log_2 \mathrm{n} \rceil + 1$, we have $\lceil \log_2 \mathrm{n} \rceil + 1 \leq N_S \leq 2 \cdot \lceil \log_2 \mathrm{n} \rceil + 1$. 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.

TABLE III
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 $m \geq 1$
see listing below	see listing below

```
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;
```

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

Proof: From Algorithm BI_basic, we have $\lceil \log_2 \mathtt{n} \rceil + 1 \leq N_S \leq 2 \cdot \lceil \log_2 \mathtt{n} \rceil + 1$. 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) $H(x) \leq \log_2 \mathtt{n}$ and 2) $H(x) = \log_2 \mathtt{n}$ 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. 3(b) 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 III-BI_v1. The BI_v1 algorithm, a recursive algorithm, is not necessarily faster than the sequential scanning; for example, in Fig. 3(b), we have $N_S = 11$, which is more than $n = 8$. We define two nodes as siblings if and only if they share a common parent.

Lemma 1: For algorithm BI_v1, when $N_I = 1$ and Q is unknown (i.e., $m \geq 1$), we have the following: The lower bound of N_S is $|N_S|_{\min} = 2\lceil \log_2 \mathtt{n} \rceil + 1$, and the lower bound can be achieved only if $m \leq 2$ 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 $N_S = 2 \cdot \lceil \log_2 \mathtt{n} \rceil + 1$. If $\mathrm{m} = 2$ and if the two dirty nodes are siblings, then $N_S = 2 \cdot \lceil \log_2 \mathtt{n} \rceil + 1$ still holds because no extra probe is needed. If $\mathrm{m} = 2$ and if the two dirty nodes are not siblings, then the probing process will fork before it reaches leaves, which cost additional probing steps. If $\mathrm{m} > 2$, 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. 2). Therefore, after adding one leaf node, the tree size is $2 \cdot k - 1 + 2 = 2 \cdot k + 1$, and our claim still holds. Since each node could be a 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; this decreases N_S. Therefore, the proof of the second part completes. ■

From Lemma 1, we know that BI_v1 can be worse than the scanning approach in some cases. 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 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 has been been 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. 3(b), 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 has been done for all levels.

Lemma 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. Considering level i, we write the following $N_S = N_S^i + (N_S^1 + \ldots + N_S^{i-1} + N_S^{i+1} + \ldots N_S^h) = N_S^i + N_{rest}$, 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 for 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 are 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. 3(b), 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 $\mathrm{N_S}$.

Lemma 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 3 is omitted because the proof is similar to the proof of Lemma 2 except that every time a one-round depairing is finished, N_S will be increased at least by 2.

Lemma 4: For algorithm BI_v1, given an arbitrary binary inspection tree with m dirty leaves among n leaves, we have

$$|N_S|_{\min} = \begin{cases} 2\cdot(m+h)-2g(m)-1 & m=2k-1, k=1,2,\ldots,\frac{n}{2}. \\ 2\cdot(m-1+h)-2g(m-1)-1 & m=2k, k=1,2,\ldots,\frac{n}{2}. \end{cases}$$

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

Proof: 1). Given $\mathrm{P1} = [\mathrm{v_1}, \mathrm{v_2}, \ldots \mathrm{v_m}, \mathrm{v_{m+1}} \ldots, \mathrm{v_n}]$ such that meters from $\mathrm{v_1}$ to $\mathrm{v_m}$ 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 $|\mathrm{N_S}|_{\min}$ based on P1. Let $f(m,h)$ 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 is an odd number, $2^i \leq m < 2^{i+1}$ and h is the tree height, i.e., $h = \lceil \log_2 \mathrm{n} \rceil + 1$. The tree consists of three parts (see Fig. 4(a)): $f(2^i, i+1)$ 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 can 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 before it reaches part 1 and part 2, it takes 1 probe (the root node) and 2 probes for other levels (if 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:

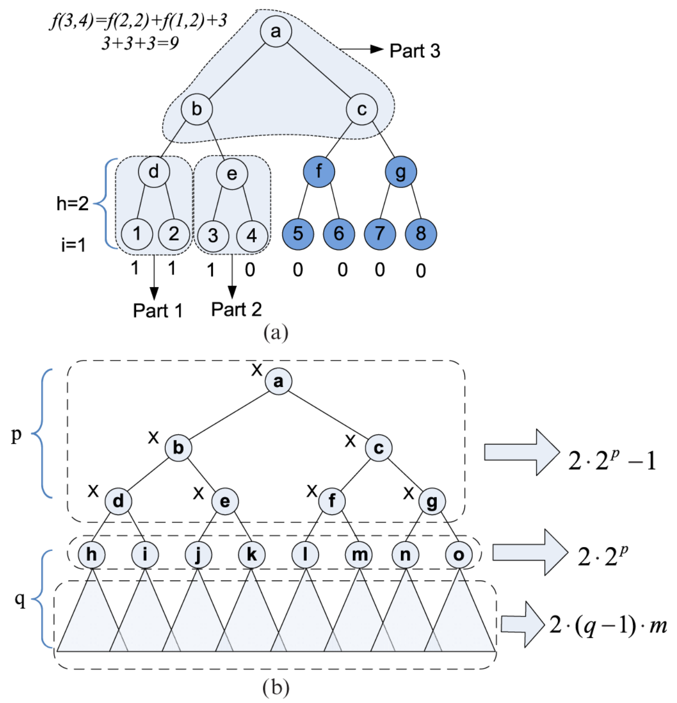


Fig. 4. Example of meter arrangement. (a) An example of the best case; (b) The worst case.

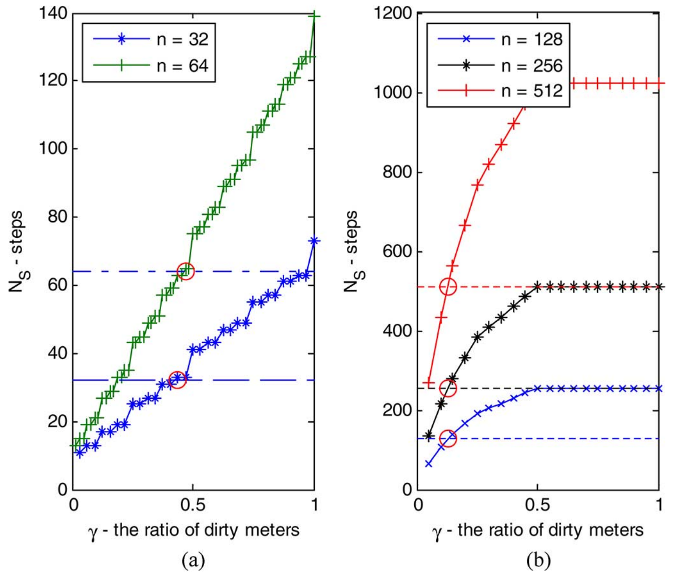


Fig. 5. Numeric results of BI_v1. (a) lower bound; (b) upper bound.

$f(2^i, i+1) = 2^{i+1} - 1$, $f(0,h) = 0$, $f(1,h) = 2h - 1$, and $f(2x,h) = f(2x-1,h)$, $x = 1,2,3,\ldots$, and i $= 0,1,2,\ldots$. By solving the recurrence with the recurrence tree method [16], we have $f(m,h) = 2\cdot(m+h) - 2g(m) - 1$. Fig. 4(a) shows an example of $f(3,4)$.

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. 4(b), 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. 4(b), 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 p level 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 $p + q = \lceil \log_2 \mathtt{n} \rceil + 1$, $p = \lceil \log_2 \mathtt{m} \rceil + 1$. Combining the former equations, we have

$$N_S = 2\cdot 2^{\lceil \log_2 m \rceil} - 1 + 2\cdot(\lceil \log_2 \mathtt{n} \rceil - \lceil \log_2 m \rceil)\cdot m$$

■

We compare BI_v1 and scanning in Fig. 5. For the best case (i.e., lower bound), BI_v1 will be better than scanning when the dirty node ratio is less than 0.5. For the worst case (i.e., upper bound), BI_v1 will be better than scanning when the dirty node ratio is less than 0.13.

TABLE IV
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;
```

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 an available inspector. If we let N_S be a function of N_I, then the exact number of steps is

$$N_S(N_I) = \left\lfloor \frac{N_S(1)}{N_I} \right\rfloor + N_S(1) \mathrm{mod} N_I.$$

C. 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 clean, then *node.rChild* is definitely dirty and can be skipped to save steps. For example, in Fig. 3(b), 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. BI_V2 is given in Table IV.

D. 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; 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 binary tree inspection and scanning. Jumping range means the number of levels skipped by jumping, which is termed by D-Jumping; 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; 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 an Adaptive Tree Inspection (ATI) algorithm. The ATI method is based upon one claim: in a binary inspection tree, sub-trees in the same level have similarity 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 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 as a dirty node. 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, the higher R is, the higher is the probability of jumping will be.
- 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:

a) when q is a leaf or the parent of a leaf, Jumping is unnecessary since it is meaningless;

b) if $R < 0.13$, Jumping is not applied to q; the value 0.13 is from our analysis result in lemma 2.

c) if $R \geq 0.13$, Jumping is applied. The jumping range D is determined by both R and S_l. In order to make sure that $1 \leq D \leq h - l - 1 = \lceil \log_2 \mathtt{n} \rceil - l$, we define $D = \lceil \lceil \log_2 \mathtt{n} \rceil - l - (S_l/R) \cdot (\lceil \log_2 \mathtt{n} \rceil - l) \rceil$. 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 help.

TABLE V
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, $h = \lceil \log_2 \mathrm{n} \rceil + 1$,
$N[1{:}h{-}1]=0$, $S[1:h-2] = +\infty$, $Q := \varnothing$, 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)
            $Q := Q \cup \{node\}$
            numMeter++;
            numDirtyMeter++;
            R = numDirtyMeter / numMeter;
            node.subR = 1;
            //update
            updateAncestor(node);
        else // internal node
            $N_{\mathrm{node.level}}$++;
            D = decide(R, node.level, $S_{\mathrm{node.level}}$);
            destNode := node;
            // jumping, if needed
            for(i=0; i <= D; ++i)
                destNode := node.lchild;
                $N_{\mathrm{destNode.level}}$ += $2^{i+1}$;
            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 += $2^{(h\text{-node.level})}$
            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)
            $S_{\mathrm{tmp.level}} = \sqrt{\frac{1}{N_{tmp.level}-1} \cdot \sum_{i=1}^{N_{tmp.level}} (i.subRatio - R)^2}$
// end updateAncestor

updateDecendent(tmp):
    k = 0;
    while(tmp.level <= h-2)
        $N_{\mathrm{tmp.level}}$ += $2^k$;
        $S_{\mathrm{tmp.level}} = \sqrt{\frac{1}{N_{tmp.level}-1} \cdot \sum_{i=1}^{N_{tmp.level}} (i.subRatio - R)^2}$
        tmp := tmp.lchild;
        k++;
//end updateDecendent
```

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 suggestive. 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 given in Table V:

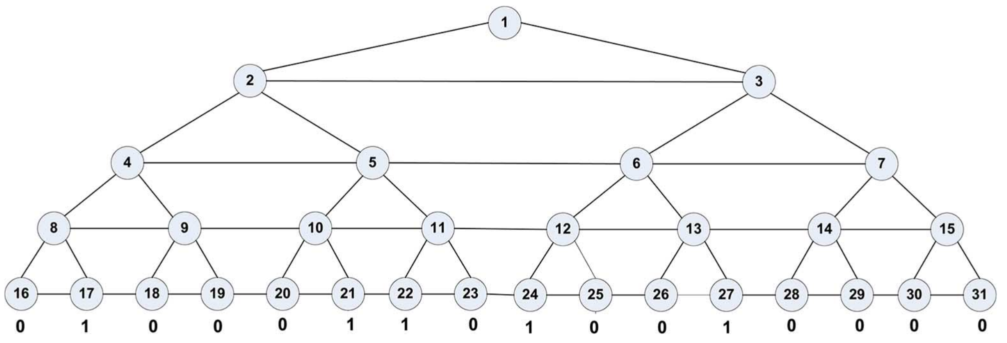


Fig. 6. An example of ATI.

TABLE VI
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	--

An example of ATI is given in Fig. 6. With a permutation of [0100011010010000], we can track the process with Table VI. It is shown that when inspection reaches nodes 5, 3, and 6, decision is made about jumping. 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 with BI_V1. However, when the scale gets large, ATI will be more advantageous.

V. DYNAMIC INSPECTION

When set Q is dynamic, things will become more complicated. For example, a previously checked meter may become malicious and cause interference in 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 $(N_I \geq 1)$ inspectors.

TABLE VII
SCANNING ALGORITHM FOR DYNAMIC INSPECTION (D-SCANNING)

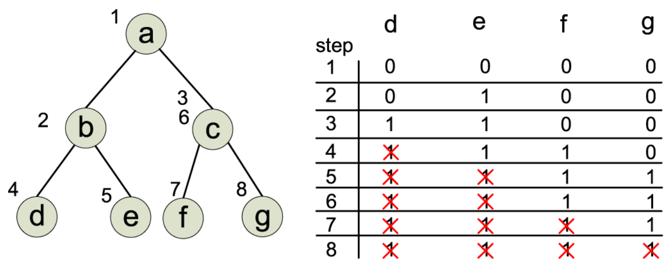
```
Algorithm: D-Scanning
Input: user set $U$ with size $k$, inspector set $I$
Output: malicious meter set $Q$
Initial: $Q := \varnothing$
DI_Scan($U$):
    if ($U$ is empty)
        return $Q$;
    while (there is unchecked meter in $U$)
        for each inspector in $I$
            if (there is unchecked meter in $U$)
                take an unchecked meter $m_i$
                if (probe($m_i$) == 'dirty')
                    $U := U - \{m_i\}$
                    $Q := Q \cup \{m_i\}$
                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($U$) $\cup$ $Q$          // start a new round
```

A. 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 VII. 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 global monitor 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.

B. 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. 7 shows an example of dynamic inspection. It can be seen that the dirty meter



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

Fig. 7. An example of dynamic binary-tree-based inspection with one inspector.

TABLE VIII
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 ;
```

set Q has been growing from $\emptyset$ 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 order to identify the meters that may be potentially dirty in future steps. 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 1st round. After removing d and e from the user set, the 2nd 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 VIII. 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.

C. 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 upon 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.

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 is not a 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 IX 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 IX. We have

TABLE IX
TRACKING INSPECTION PROGRESS WITH SCANNING APPROACH

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

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

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

Lemma 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 $|N_S|_{\max} = \lceil (1/N_I) \cdot (m \cdot n - (1/2) \cdot (m^2 - m)) \rceil$, and this bound can be achieved if and only if $x_i = 1$ for all $i = 1, 2, \ldots, r$, (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 $|N_S|_{\min} = \lceil (1/N_I) \cdot (2 \cdot n - (m - 1)) \rceil$, 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 $N_I = 1$. Based upon (1) and Table IX, 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 $i = 1, 2, \ldots, r$, the size of set S is maximal, and S becomes $S_0 = \{n, n - 1, n - 2, \ldots, n - (m - 1)\}$. 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.

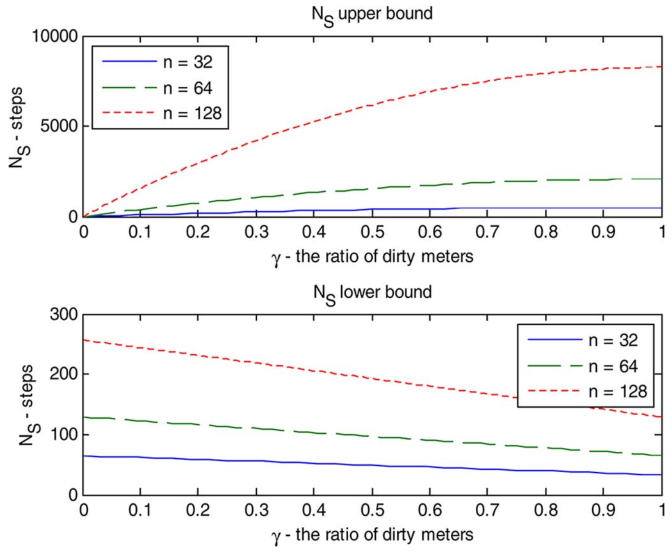


Fig. 8. the upper bound (top) and lower bound (bottom) of N_S with scanning approach when $N_I = 1$.

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 $\{n, n-6, n-7, \ldots, n-(m-1)\}$; 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 decrease N_S, therefore $|N_S|_{\max} = \sum_{t=0}^{m-1}(n-t) = m \cdot n - (1/2)\cdot(m^2 - m)$; 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 $N_I = 1$, $|N_S|_{\max} = m \cdot n - (1/2)\cdot(m^2-m)$, 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. ■

The numeric results of the bounds of N_S with scanning approach are shown in Fig. 8.

VI. Evaluation

A. 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 bound 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. 9 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 walking the tree based upon the heuristic information.

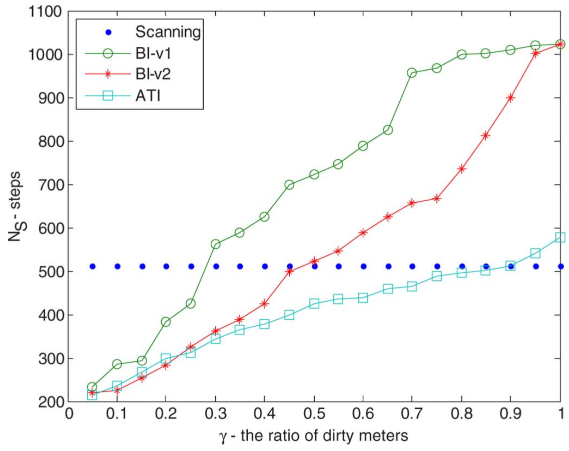


Fig. 9. Evaluation results when $n = 512$.

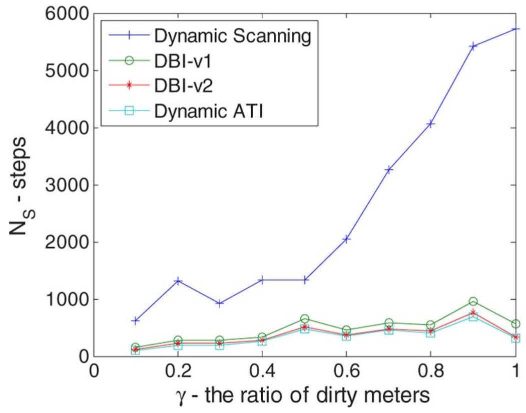


Fig. 10. The average performance of DBI_v1,DBI_v2, D-scanning, and D-ATI when $n = 128$.

B. Dynamic Inspection

We have determined the bounds of N_S when different approaches are employed. However, the dynamic inspection is featured by 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. 10, 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 fix m 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

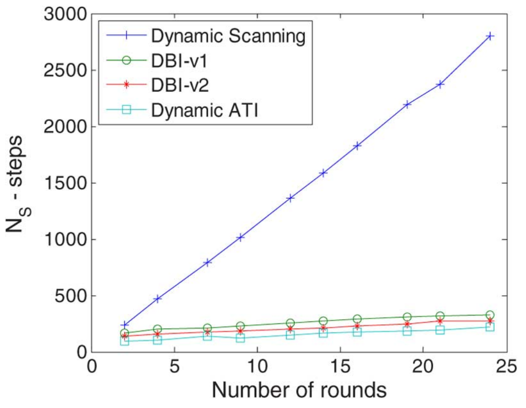


Fig. 11. The number of rounds Vs. the number of steps ($n = 128$, $m = 24$).

optimization in DBI_v2 happens with a condition and 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 this real world situation, scanning has already lost its advantage.

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. 11 in which the curves are all climbing. For a specific r, the tree-based inspection schemes are far better than scanning; especially when r is getting larger, the performance gap between the tree-based inspection and scanning is 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 causing the undermining of the efficiency of te scanning approach, which only outperforms the tree-based inspection when m is sufficiently large.

VII. Conclusion

In this paper, we have discussed the malicious meter inspection problem in neighborhood area smart grids. We have formulized the MMI problem and analyzed the solution algorithms that are based upon an inspection tree. We have thoroughly studied and compared the performance of each algorithm. We have also proposed an adaptive tree inspection scheme that employs 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. Our theoretical and numeric results have shown the effectiveness of this approach.

References

[1] J. Liu, Y. Xiao, and J. Gao, "Accountability in smart grids," in *Proc. IEEE CCNC 2011*, pp. 1166–1170.

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

[3] C. Shuyong, S. Shufang, L. I. Lanxin, and S. Jie, "Survey on smart grid technology," *Power Syst. Technol.*, vol. 33, no. 8, pp. 1–7, 2009.

[4] H. Farhangi, "The path of the smart grid," *IEEE Power Energy Mag.*, vol. 8, no. 1, pp. 18–28, 2009.

[5] 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.

[6] M. Amin and B. F. Wollenberg, "Toward a smart grid: Power delivery for the 21st century," *IEEE Power Energy Mag.*, vol. 3, no. 5, pp. 34–41, 2005.

[7] P. McDaniel and S. McLaughlin, "Security and privacy challenges in the smart grid," *IEEE Security Privacy*, vol. 7, no. 3, pp. 75–77, 2009.

[8] Z. Xiao, Y. Xiao, and D. Du, "Building accountable smart grids in neighborhood area networks," in *Proc. IEEE GLOBECOM*, 2011.

[9] Electric Light and Power Magazine "Reducing revenue leakage," [Online]. Available: http://uaelp.pennnet.com/ 2009

[10] S. S. S. R. Depuru, L. Wang, and V. Devabhaktuni, "Support vector machine based data classification for detection of electricity theft," in *Proc. 2011 IEEE/PES*, pp. 1–8.

[11] M. Madrazo, "Today's energy theft detection models help protect revenues while enhancing neighborhood safety," [Online]. Available: http://www.pipelineandgasjournal.com/today's-energy-theft-detection-models-help-protect-revenues-while-enhancing-neighborhood-safety Jul. 2010, vol. 237, no. 7

[12] S. McLaughlin, D. Podkuiko, and P. McDaniel, "Energy theft in the advanced metering infrastructure," Critical Information Infrastructures Security, Lecture Notes in Computer Science 2010, vol. 6027/2010, pp. 176–187, 10.1007/978-3-642-14379-3_15.

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

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

[15] 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," in *Proc. 2003 IEEE PES Transmission Distribution Conf. Expo.*, 2003, vol. 1, pp. 163–168, Vol.1.

[16] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. New York: Wiley, 2001, ch. 4, sec. IV.2.

[17] Y. Xiao, "Editorial," *Proc. Int. J. Security Netw., Special Issue Security Privacy in Smart Grid*, vol. 6, no. 1, pp. 1–1, 2011.

[18] D. Kundur, X. Feng, S. Mashayekh, S. Liu, T. Zourntos, and K. L. Butler-Purry, "Towards modelling the impact of cyber attacks on a smart grid," *Int. J. Security Netw.*, vol. 6, no. 1, pp. 2–13, 2011.

[19] G. Kalogridis, S. Z. Denic, T. Lewis, and R. Cepeda, "Privacy protection system and metrics for hiding electrical events," *Int. J. Security Netw.*, vol. 6, no. 1, pp. 14–27, 2011.

[20] F. Li, B. Luo, and P. Liu, "Secure and privacy-preserving information aggregation for smart grids," *Int. J. Security Netw.*, vol. 6, no. 1, pp. 28–39, 2011.

[21] J. Zhang and C. A. Gunter, "Application-aware secure multicast for power grid communications," *Int. J. Security Netw.*, vol. 6, no. 1, pp. 40–52, 2011.

[22] 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*, DOI: 10.1002/sec.698, accepted for publication.

[23] Z. Xiao, Y. Xiao, and D. Du, "Non-repudiation in neighborhood area networks for smart grid," *IEEE Commun. Mag.*, vol. 14, no. 4, pp. 981–997, 2012.

[24] J. Liu, Y. Xiao, S. Li, W. Liang, and C. L. P. Chen, "Cyber security and privacy issues in smart grids," *IEEE Commun. Surveys Tutorials*, 10.1109/SURV.2011.122111.00145, accepted for publication.

[25] 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 Syst.*, vol. 28, no. 2, pp. 391–404, Feb. 2012.

[26] J. Liu, Y. Xiao, and G. Gao, "Achieving accountability in smart grids," *IEEE Syst. J.*, to be published.

Zhifeng Xiao received the B.S. degree in computer science from Shandong University, China, in 2008. He is working toward the Ph.D. degree in the Department of Computer Science at the University of Alabama.

His research interests are in design and analysis of secure distributed and Internet systems.

Yang Xiao (SM'04) worked in industry as a MAC (Medium Access Control) architect involving the IEEE 802.11 standard enhancement work before he joined Department of Computer Science at The Univ. of Memphis in 2002.

He is currently with the Department of Computer Science (with tenure) at University of Alabama. He was a voting member of IEEE 802.11 Working Group from 2001 to 2004. He serves as a panelist for the US National Science Foundation (NSF), Canada Foundation for Innovation (CFI)'s Telecommunications expert committee, and the American Institute of Biological Sciences (AIBS), as well as a referee/reviewer for many national and international funding agencies. His research areas are security and communications/networks. He has published more than 200 refereed journal papers (including 50 IEEE/ACM transactions papers) and over 200 refereed conference papers and book chapters related to these research areas .His research has been supported by the US National Science Foundation (NSF), U.S. Army Research, The Global Environment for Network Innovations (GENI), Fleet Industrial Supply Center-San Diego (FISCSD), FIATECH, and The University of Alabama's Research Grants Committee.

Dr. Xiao currently serves as Editor-in-Chief for International Journal of Security and Networks (IJSN) and International Journal of Sensor Networks (IJSNet). He was the founding Editor-in-Chief for International Journal of Telemedicine and Applications (IJTA) (2007–2009).

David Hung-Chang Du (F'08) received the B.S. degree in mathematics from National Tsing-Hua University, Taiwan, in 1974, and the M.S. and Ph.D. degrees in computer science from University of Washington, Seattle, in 1980 and 1981, respectively.

He is currently the Qwest Chair Professor at the Computer Science and Engineering Department, University of Minnesota, Minneapolis. He has served as a Program Director at National Science Foundation from 2006 to 2008. His research interests include cyber security, sensor networks, multimedia computing, mass storage systems, high-speed networking, database design and CAD for VLSI circuits. He has authored and co-authored more than 240 technical papers, including 110 referred journal publications in these research areas.

Dr. Du is a Fellow of Minnesota Supercomputer Institute. He is currently served on the editorial board of several journals. He has also served as Conference Chair and Program Committee Chair to several conferences in parallel processing, security, multimedia, networking and database areas. Most recently, he was the General Chair for IEEE Security and Privacy Symposium (Oakland, California) 2009, Program Committee Co-Chair for International Conference on Parallel Processing 2009, and Workshop Co-Chair and the General Chair for IEEE International Conference on Distributed Computing Systems 2010 and 2011 respectively.