THE INSTITUTE FOR SYSTEMS RESEARCH

ISR TECHNICAL REPORT 2008-11

Efficient Handling of Adversary Attacks in Aggregation Applications

Gelareh Taban, Virgil D. Gligor

Efficient Handling of Adversary Attacks in Aggregation Applications

Gelareh Taban[1] * and Virgil D. Gligor[2]

[1] ECE, University of Maryland, College Park, gelareh@umd.edu
[2] ECE and CyLab, Carnegie Mellon University, gligor@cmu.edu

Abstract. Current approaches to handling adversary attacks against data aggregation in sensor networks either aim exclusively at the detection of aggregate data corruption or provide rather inefficient ways to identify the nodes captured by an adversary (e.g., O(cn) messages, in the worst case, for detecting c captured nodes in a network of n nodes). In contrast, we propose a distributed algorithm for efficient identification of captured nodes; i.e., O($c\ log^3 n$) over a constant number of rounds. Our algorithm does not assume a fixed upper bound on the number of captured nodes. We formulate our problem as a combinatorial group testing problem and show that this formulation leads not only to efficient identification (and subsequent removal) of captured nodes but also to a precise cost-based characterization of when in-network aggregation retains its assumed benefits in a sensor network operating under persistent attacks (i.e., where an adversary continues to corrupt data aggregates until captured nodes are identified).

1 Introduction

Data aggregation is generally believed to be a fundamental communication primitive in resource-constrained, wireless sensor networks. In principle in-network aggregation of sensor data can drastically reduce network communication. To accomplish this, nodes are organized as a tree – called the aggregation tree – that is rooted at a Base Station (BS). In response to BS queries, nodes aggregate the critical data they receive from their descendents along with their own data, and forward the partial aggregate to their ancestor node in the aggregation tree.

Motivation. An undesirable effect of aggregation is that an aggregator node that is captured by an adversary could report arbitrary values as its aggregation result, thereby controlling not only its own measurements but also that of all the nodes in its entire aggregation sub-tree. As a consequence, an adversary who can

* This research was supported in part by US Army Research Laboratory and the UK Ministry of Defence under Agreement Number W911NF-06-3-0001 and by the US Army Research Office under Contract W911NF-07-1-0287 at the University of Maryland. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official poli- cies, either expressed or implied, of the US Army Research Laboratory, US Army Research Office, the U.S. Government, the UK Ministry of Defense, or the UK Government.

capture nodes selectively (e.g., close to the base station) could corrupt the entire network aggregation process, while incurring minimal cost and effort. Therefore, to achieve reliable aggregation, and, in particular, to assure the integrity of aggregation process it is important (i) to detect the adversarys presence in the network (i.e., by discovering aggregated-data corruption) and (ii) to identify and remove (e.g., revoke) the captured nodes that corrupt data aggregates.

Most recent work on secure data aggregation has focused exclusively on efficient *detection* of integrity breaches in the aggregation process by captured nodes (e.g. [8, 2, 12, 9, 6]). While detection of integrity breach is the first necessary step in achieving secure data aggregation, it does not provide a fully adequate response to malicious-node behavior; i.e., detection of integrity breaches alone does not unambiguously identify and remove malicious nodes from the network. Any countermeasure against an aggregation adversary based exclusively on detection of corrupt aggregate results would leave the network unprotected against repeated attacks that deny service to the base station. An effective approach to handling this problem would (i) identify corrupted nodes and remove them from the aggregation process (e.g. node revocation), and (ii) ensure continued, but gracefully degraded aggregation services, even during an attack period. Identification and removal of corrupted nodes have the added benefit of acting as a deterrent against some potential adversaries who might avoid the risk of being identified.

Problem. We consider an aggregation scenario where a subset of nodes is corrupted by an adversary. A corrupted node can (i) insert a false data into the network or (ii) if it is an aggregating node, output a false aggregation result. The goal of the corrupted node is to convince the BS to accept an invalid value. Since the network cannot protect against the insertion of incorrect data into the network without assuming specific distributions on the environmental data [12, 11], we simply assume that all valid sensor inputs r must be within a given range $r_1 < r < r_2$. Our objective is to (i) detect an attack in the network, (ii) identify malicious nodes, (iii) ensure graceful degradation of the aggregate with respect to the number of corrupted nodes in the network, while (iv) retaining the *efficiency advantages* of data aggregation.

A straight-forward method of achieving the first three stated objectives *without retaining in-network aggregation*, henceforth called the *baseline scheme*, would be to detect the presence of malicious behavior in the network [6, 2], and then require each aggregate node to directly transmit their data without aggregation along with a message authentication code (MAC) to the BS. By excluding in-network aggregation, we would trivially remove any attacks on the aggregation process. The BS could then identify any malicious nodes that inject a false data by range testing the received data. If the corrupted nodes are persistently malicious, the BS could identify *all* corrupted nodes. Furthermore, the BS itself could reconstruct the network aggregate by disregarding the data of all malicious nodes and finally guarantee the correctness of this network aggregate based on the security of the MAC protocol and the data validity verification. Although the baseline scheme would satisfy the first three objectives mentioned

above, it would do so at the cost of removing in-network data aggregation and the associated communication efficiency. For this reason, we do not consider the baseline scheme to be a useful solution. Nevertheless, we use the baseline scheme as a practical lower bound on the performance of any secure aggregation solution that satisfies our three objectives above. That is, any efficient solution must have better performance than the baseline scheme; otherwise, the baseline scheme becomes preferable, and the entire notion of in-network data aggregation ceases to be useful in hostile environments.

Related Work. We first focus on existing literature for integrity verification of in-network data aggregation and then briefly introduce the field of group testing.

In-network Aggregation. Hu and Evans [8] propose aggregation verification through re-computation of the aggregate by up-stream nodes (essentially centralized verification using a sliding window). This approach is effective for a limited range of attacks as it assumes an upper bound t on the number of corrupted nodes in the system. An adversary can escape detection by compromising more than t nodes. At the limit, as the bound t approaches the size of the network n, the communication cost incurred approaches that of the baseline scheme and in-network data aggregation ceases to be useful.

Yang et al. [12] propose a scheme that probabilistically divides the network into aggregating groups and considers a group suspicious if its aggregate is an outlier. This approach requires the restrictive assumption that an event is globally visible across the network and generates similar readings in all nodes. In practice, however, most events and monitoring regions are localized (e.g. bushfire detection or perimeter monitoring). Furthermore, suspicious aggregates are attested via centralized verification, therefore incurring communication costs in the order of network size.

Chan et al. [2] propose a fully distributed aggregation verification algorithm, called the Secure Hierarchical In-network Aggregation (SHIA), which detects the existence of any misbehavior in the aggregation process. The scheme perfectly satisfies its objective as a detection mechanism; however the protocol design is not intended to address our problem as it does not aim either at the identification and removal of adversary nodes, or at providing continuous, but gracefully degraded, service under attack. Similarly, the work of Frikken and Dougherty [6], which improves the performance of SHIA, aims only at the detection of attacks against the aggregation process.

Haghani et al. [7] extend SHIA to allow the identification of malicious node once an attack is detected. A misbehaving node is detected via successive polling of the layers on a commitment tree (generated during the aggregation process) by the base station. Although this work is closest to ours in spirit, it is different in the following three fundamental ways. First, it incurs high communication cost as it not only relies on centralized identification but also each run of the algorithm identifies only one malicious node. In the worst case, to detect c malicious nodes in a network of size n, $\mathcal{O}(nc)$ messages are generated per link. In contrast the cost of our scheme is logarithmic in n over a constant number of rounds. Second, the performance analysis and adversary model presented excludes a comparison

with the baseline scheme (where identification of adversary nodes incurs a cost of only $\mathcal{O}(n)$) as the adversary attacks only the aggregation process. Hence, it is not clear at what point the proposed scheme ceases to be useful. Finally, [7] does not provide network service during the period of the attack.

Group Testing: The identification of corrupted nodes is directly related to the problem of group testing, which aims to identify the defective items of a given set through a sequence of tests. Each test is on a subset of items and indicates whether the subset contains a defective item. In combinatorial group testing, there is a constant number of defectives in a set. This number can be either known or unknown at the time of testing. Group testing is efficient when the number of defectives in a sample space is small compared to the total number of samples [4]. This is an analogous setup to untrusted sensor networks which are characterized as large, densely packed network of sensor nodes.

Our Contributions. We propose a divide-and-conquer approach to trace and remove malicious nodes from the network that achieves the four objectives stated above. At a high level, the approach recursively (i) partitions the suspicious subset of the network, (ii) runs a 'test' in each partition to check the correctness of the sub-aggregation value, (iii) if the result is bad, the set remains suspicious, else it is considered good and the associated sub-aggregate value is retained. Hence, the algorithm allows for the incremental reconstruction of lost data from sub-aggregated value, over the course of its execution. The algorithm terminates when it has isolated all the malicious nodes in the network. The partition test is a secure aggregation primitive; we show the requirements for this primitive and present a modified version of SHIA [2, 6] with relaxed assumptions for this primitive. The identification algorithm is designed and optimized with respect to the communication cost for an arbitrary number of malicious nodes. We prove the correctness of the algorithm and evaluate its performance using an analysis method inspired by the field of combinatorial group testing [4]. Our results illustrate the relationship between the efficiency of malicious-node identification and the number and distribution of these nodes. In particular, we define a precise cost-based threshold when in-network data aggregation ceases to be useful in hostile environments.

2 Preliminaries

System Model. Consider a multihop network of untrusted sensor nodes and a single trusted base station. The system administrator or user that resides outside the network interacts with the network through the BS interface. For brevity, subsequently we refer to any requests made by this external entity via the BS, as simply requests by the BS.

We assume that each sensor has a unique identifier v and a unique secret key shared with the base station, K_v. The sensor network continuously monitors its environment and measures some environmental data. We divide time into epochs; during each time epoch, the BS broadcasts a data request to the nodes

in the network and nodes forward their data response back to the BS. Data can be forwarded individually or as an aggregate.

We model node corruption in the network as a function of the number c and the distribution of the corrupted nodes. Each sensor node v belongs either to the good set G or the malicious/corrupted set M. A network instance is defined as $N = \{\forall v \text{ in network} : v \in G \vee v \in M\}$ where $|M| = c$ and $G = N \setminus M$. The collection of all N for a given c, constitutes a family of networks $\mathcal{N}_c$.

For the purpose of computing the aggregate, we assume that the sensed environment (e.g. temperature or humidity) changes minimally with respect to the duration of the identification algorithm. This is a practical assumption as once malicious activity is detected, the identification algorithm is promptly executed. Moreover the algorithm terminates after a small, constant number of rounds (logarithmic in network size).

Adversary Model. We assume that the network is deployed in an adversarial environment where the adversary can corrupt an arbitrary number of nodes. Once a node is corrupted, the adversary has total control over the secret data of the node as well as the subsequent behavior of the sensor node. We assume that a corrupted node *persistently* misbehaves by inducing the base station to accept an 'illegal' value. An illegal value is defined based on the adversary objectives which is to induce the BS to accept a data value which is not already achievable by direct data injection. A direct data injection attack occurs when an adversary modifies the data readings reported by the nodes under its direct control, under the constraint that only legal readings in $[r_1, r_2]$ are reported [2]. In the case of a single data values, this means that the data value transmitted is outside the legal reading of $[r_1, r_2]$. We call this a *false data injection attack*.

In the case of data aggregation, the objective of the adversary is to tamper with the aggregation process such that the BS accepts an aggregation result which is not achievable by the direct data injection. We refer to this type of attack as a *false aggregation attack*. An aggregation protocol is considered secure if the adversary cannot successfully launch such an attack [2].

It is possible to reduce our adversary to only attack the aggregation process (similar to [7]). Although our protocol can be adapted to this model[3], an important feature of our model is that it allows comparison with the baseline scheme.

Performance Measure. We use link cost as a metric to analyze our algorithm. Link cost is defined as the total number of messages transmitted over a particular link in the network. Node congestion can be derived as a function of worst link cost and the maximum node degree in the network. Link cost is a commonly used measure in sensor and ad hoc networks as it determines how quickly nodes in the network exhaust their energy supply. Such nodes are often core to the connectivity or the functionality of the network and their loss can lead to network partitioning or denial of service.

[3] This can be done by forcing the algorithm to terminate when suspicious partitions are of size two and revoking both nodes in the group.

3 Identification Algorithm

The main objective of our algorithm is to recursively isolate the malicious nodes in the network and thus render the adversary inoperative. The algorithm is initiated once misbehavior is detected in the network (e.g. via [2]) and is executed over a number of rounds, following an intuitive divide-and-conquer approach. In each round the algorithm partitions the suspicious subsets of the network and performs a partition test (defined in next section) on the newly formed groups. The number of subsets a suspicious group is partitioned into is called the *partition degree*. The partition test consists of nodes aggregating their data and verifying the integrity of their aggregation process. The test has two outputs: 'pure' if all the nodes in the partition are good and 'impure' if there is at least one malicious node in the group. The algorithm terminates when there are no more remaining impure groups.

By distributing the localization of the malicious nodes, the scheme simply keeps track of the lower bound on the number of malicious nodes in the network and increases the bound only when the findings of the scheme up to that point imply that this is valid.

Algorithm 1 ***Identification***

Input: *All the nodes in the network $N \in \mathcal{N}_c$, partition degree $m > 1$, where m is an integer, representing the number of partitions a group divides into in each iteration.*

Output: *A result set M of malicious nodes and a result set G of good nodes, such that $M \cup G = N$*

Let $t = 1$ be the lower bound on the number of malicious nodes in the network and $S = \cup_{i=1}^{t} S_i$ denote the current set of suspicious nodes, $S_1 = N$.

1. For $j = 1, \cdots, t$, BS requests partition S_j to be divided into m disjoint partitions (using partition rule algorithm 2). The collection of subdivided sets form the current collection S. Set t to be the cardinality of set S.

2. For $j = 1, \cdots, t$, if $|S_j| > 1$, the nodes in partition S_j partition themselves into groups of size $\frac{n}{m}$ and execute partition test. BS verifies the purity of each partition.

3. The BS learns the status of each node for the following round (details are provided in the next section). For $j = 1, \cdots, t$, if S_j is pure (i.e. all the nodes are good), then $G = G \cup S_j$; else if S_j is impure (i.e. there is at least one misbehaving node) and a singleton set, then $M = M \cup S_j$ and decrement t. Adjust the indices of the remaining sets, $\{S_j\}$ appropriately, to include only sets that are impure and non-singleton. If $t > 0$, go to step 1 (next round), else quit as all malicious nodes have been traced.

We can model the divide-and-conquer approach of algorithm 1 as the pruning process of an m-ary tree T where each tree vertex is associated with a partition test. The root of tree T is associated with the input set N and each round i

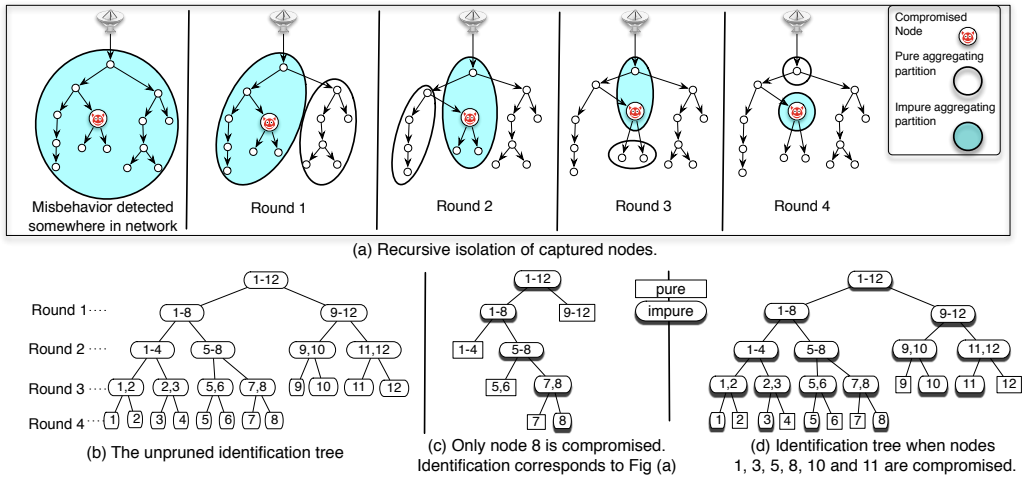


Fig. 1. Identification algorithm on an input of 12 nodes, $m = 2$.

is associated with level $(i + 1)$ of the tree. This is because the identification algorithm is initiated when misbehavior is detected in the network and therefore the test at level 1 has been already executed. If a partition X is tested pure, then all the descendants of the associated vertex are pruned; otherwise the set X is re-partitioned. Figure 1(b) presents an unpruned identification tree for a network of 12 nodes and partition degree $m = 2$. Figures 1(c) and (d) show how the tree can be pruned when the network contains one and six corrupted nodes respectively. Figure 1(a) shows how the identification tree corresponds to the recursive isolation of the captured nodes on the physical network.

Next we define a novel partition rule inspired by [3]. This algorithm partitions the network such that the identification tree contains at most one incomplete subtree. Intuitively a complete tree of n nodes executes less or equal number of tests than an incomplete tree of n nodes as the complete tree contains less vertices (where each vertex corresponds to one test).

Algorithm 2 ***Partitioning Rule***

Input: *Set X, maximum number of partitions m.*

Output: *Result sets $\{X_i\}$, such that $\cup X_i = X$.*

Let $i = 1$ denote the new subset (X_i) to be determined.

1. Choose X_i to contain $m^{\lceil \log_m |X| \rceil - 1}$ nodes.

2. Update set $X = X \setminus X_i$ to exclude the newly formed subset. If less than m subsets have been formed and X contains more than $m-1$ nodes, then increment i and go to Step 1.

Else if X is not a singleton set, increment i and let X_i include the remaining nodes in X.

Else if X is a singleton set, then X cannot be partitioned anymore.

3.1 Partition Test

The test that nodes perform in each newly formed partition is a fundamental step in our algorithm. There are two types of tests depending if the partition is a singleton or otherwise.

Tests for Non-singleton Partitions In all non-singleton partitions (that is, partitions which contains more than one node), data is aggregated and the partition leader directly transmits (via multi-hop) the partition aggregate to the base station. The base station then verifies the integrity of the aggregation process and hence the integrity of the nodes within the partition. Algorithm 1 can be composed with any aggregation-verification algorithm that does not depend on a fixed partition and provides provable guarantees (e.g. modified versions of [2] or [8]). In particular an administrator can interchange algorithms and choose the most appropriate algorithm for varying settings and assumptions. In this work, we propose and use a modified version of SHIA [6, 2]. Before we present our extension, we will first provide an overview of SHIA and describe how we modify the algorithm.

SHIA extends the aggregate-commit-prove framework of [9]. In the aggregate-commit phase of the algorithm, a cryptographic commitment tree (hash tree) is built based on the sensor readings and the aggregation process. This forces the adversary to choose a fixed aggregation topology and set of aggregation results. In the prove phase of the algorithm, each sensor independently verifies that the final aggregate has incorporated its sensed reading correctly. Specifically each sensor reconstructs the commitment structure and ensures that the adversary has not modified or discarded the contributions of the node.

SHIA cannot be used as is because it assumes that the base station knows the exact set of nodes which are alive and reachable. We propose a new algorithm Group SHIA (GSHIA) (presented in appendix A) which includes two additional properties. First, nodes should be able to organize themselves into groups of size g, where g is arbitrarily defined by the BS. This can be easily achieved as the 'delay aggregation' approach of SHIA develops an aggregation tree one node at a time. Since the root node of the aggregation tree knows the size of its subtree, it can declare a partition complete when it has g nodes or it cannot add any more nodes to its partition.

The second property we propose, is that the base station should be able to verify the integrity of the aggregation process for a group of unknown size and membership set. This property can be implemented through the use of a Bloom filter [1] that summarizes the membership information of the partition. The BS then verifies the membership set by exhaustively searching through the possible nodes. The change we propose places most of the membership resolution burden on the BS, which is generally assumed to be powerful. However we can reduce the computation burden by noting that algorithm 1 is *nested* (i.e. each new partition is a proper subset of an older impure partition) and therefore the space of possible partitions in each round is reduced by a factor of m. Further

improvements can be made if the base station knows the topology of the network a priori, using efficient schemes such as [10].

We can also improve the communication burden that is placed on the nodes due to the Bloom filter by noting that the BS does not require the filter output of every partition (although the partition tag, defined in appendix A, is needed for every partition). For example if a given partition is halved, the BS only needs to know the filter output of only one of the new partitions; the other partition can be trivially resolved. More exact knowledge of the topology information can also reduce the need for the filter output.

Tests for Singleton Partitions If a partition contains exactly one sensor node, the node v transmits its measured data x_v along with a MAC tag σ_v computed using K_v. Upon receiving $\langle v, x_v, \sigma_v \rangle$, the base station verifies the tag and ensures that x_v is valid. The base station assumes node v has misbehaved if x_v is not in the correct range but the tag verifies correctly.

3.2 Computing Aggregate

An important feature of our algorithm is that the network aggregate can tolerate malicious nodes and in fact, the aggregate degrades gracefully with the attack. In particular, dual to the intuition that the algorithm recursively isolates the corrupted nodes, is that the algorithm also increasingly identifies the uncorrupted nodes in the network. The BS can then use the data from the nodes determined to be uncorrupted to reconstruct the network service.

Recall our assumption that the sensed environment of the network changes slowly with respect to the identification algorithm. Since the algorithm converges very fast (e.g. in a network of 1000 nodes, it converges in at most 10 rounds), we can assume that the sensed environment is in fact constant over the lifetime of the algorithm. This allows us to improve the quality of the network aggregate in each successive round by incorporating the aggregates of pure groups. The amount that the aggregate improves in each round is dependent on the number of malicious nodes in the network as well as the degree of the decision tree. In algorithm 3 we present the aggregate update algorithm when the aggregation function is addition. We can easily extend this to other low-order statistics functions, such as min/max, averaging, etc.

Algorithm 3 ***Aggregate Update in Round*** i
Input: *Aggregate* Ψ_{i-1} *from round* $i-1$*, set* $\{\Psi[j]\}$ *of the aggregates of all pure partitions from round* i*.*
Output: *Aggregate* Ψ_i *of round* i*, where* $\Psi_i = \Psi_{i-1} + \sum_j \Psi[j]$*.*

3.3 Security Analysis

In the following, we first show the correctness of the proposed algorithm and in section 4, we propose a mathematical framework to analyze the communication cost associated with providing our security solution.

Theorem 1. *Given an input set of nodes N and partition degree m, algorithm 1 outputs two resulting sets of corrupt nodes M and of good nodes G, such that $M \cup G = N$.*

- *(Completeness) If node $v \in N$ and v is corrupt, then $v \in M$, i.e. no false negatives.*
- *(Soundness) If node $v \in M$, then v is corrupt, i.e. no false positives.*

Proof. Let T be the identification tree that algorithm 1 generates. For any corrupted node $v \in N$, any vertex u in T which contains v, tests impure. This is because a corrupted node is persistently malicious and the partition test $t(\cdot)$ is perfect (i.e. the test result is always correct). Each impure vertex in T is either divided into smaller partitions if it is a non-singleton set, or is added to the set M if it is a singleton set. Since the algorithm converges when $t = 0$ or when there are no more impure non-singleton partitions, then by convergence time the algorithm must have found all corrupt nodes and added them to set M. Thus the algorithm is complete.

Additionally, the algorithm is sound since if node $v \in M$, then there exists a vertex u in identification tree T which is associated with a singleton set $\{v\}$ and that $\{v\}$ is impure. Thus v must be malicious.

Corollary 1. *Algorithm 1 isolates all c corrupt nodes within $\lceil \log_m |N| \rceil$ rounds.*

We can trivially prove this as the leaf at the highest level of identification tree T defines the round duration of the algorithm. In particular note that T is rooted at a vertex associated with node set N and the root node (at level 1) is processed in round 0. Also in each round, the active vertices are divided into m equal partitions and at most one of the partitions contains a smaller number of nodes. It is thus easy to see that T has leaves on levels $\lceil \log_m |N| \rceil + 1$ and $\lceil \log_m |N| \rceil$.

4 A Theoretical Model for Cost Analysis

In this section, we derive the cost associated with the security guarantees of the proposed protocol. We first formulate the communication cost in terms of an optimization problem. We then analytically solve this problem by introducing a novel mathematical framework, inspired by [4, 5] and evaluate our results using an example network of 4096 nodes. For a complete analysis of the problem, finally we look at the best and average case cost of the system.

The link cost of the algorithm is a function of the number of partitions that are generated in each round (referred to as *partition* cost) as well as the aggregation-verification cost of each partition (referred to as the *test* cost of each partition). It is important to distinguish between the two costs because partition cost is characterized solely by the identification algorithm, whereas test cost is a function of the aggregation-verification primitive adopted and can be improved upon. We emphasize that the total cost derived in this section are based on the use of GSHIA (apprendix A) as our primitive.

4.1 Cost Upper Bound Definition

Let N be a network instance with c corrupted nodes, $N \in \mathcal{N}_c$, input to the algorithm and let the algorithm terminate in $\tau = \lceil \log_m |N| \rceil$ rounds. Let $P(i, m, N)$ denote the number of partitions in round i where each partition is of size $T(j, m, N)$, $j = 1, \cdots, P(i, m, N)$. We formulate the total communication cost $G(m, N)$ of the algorithm as:

$$G(m, N, c) = \sum_{i=0}^{\tau} \sum_{j=1}^{P(i,m,N)} T(j, m, N) \tag{1}$$

We define *worst case* communication cost of the algorithm as the maximum cost of the algorithm for all distributions of c corrupt nodes in the network :

$$G(m, c) = \max_{N \in \mathcal{N}_c} C(m, N, c) \tag{2}$$

Finally we can define the optimization problem for our proposed identification algorithm:

$$G(c) = \min_{m>1} G(m, c) \tag{3}$$

The parameters which achieve $G(c)$ are called the minimax parameters of the identification algorithm. *The goal of the network administrator is to find the minimax parameter m for a given network N without knowing the number of corrupt nodes c.*

The primary parameter in equation 3 is partition degree m. Towards solving equation 3 we consider the effect of m on the different components of cost. Test cost for singleton and non-singleton groups are O(1) and O($\log g$) (refer to appendix A) respectively, where g is the size of the group. Thus test cost is logarithmically related to $\frac{1}{m}$.

Next we present some results relating m with partition cost. This is of particular interest as our results can be applied to other divide-and-conquer algorithms. In fact the isolated problem of optimizing partition cost is equivalent to an instance of combinatorial group testing problem, where the number of defectives is unknown and we optimize the algorithm to minimize the number of tests performed. In appendix B, we present some existing results in combinatorial group testing for $m = 2$. Inspired by these results, in the following, we extend this work for the general m-ary case. To the best of our knowledge this is the first time the m-ary case has been considered.

4.2 Theoretical Results

Upper bound when n is a power of m The following theorem proves the upper bound of the partition cost for different m-ary identification algorithms, where the number of nodes in the network n is a power of m.

Theorem 2. *Let n be a power of $m > 1$. Then for c corrupted nodes in n nodes, $1 \leq c \leq n$, the number of partitions generated is tightly upper bounded by $\frac{m}{1-m} + mc(\log \frac{n}{c} + \frac{1}{m-1})$.*

Proof. Let tree T be the m-ary identification tree whose root vertex is associated with a set of size n, which is a power of m. According to the algorithm, every internal vertex must be associated with an impure set and there must exist exactly c impure leaves. We sum up the total number of pure leaves in T as follows. Let u denote the height of tree T, $u = \log_m n$. Each level i has m^{i-1} vertices, where at most c are impure. Level $v = \lceil \log c \rceil$ is the first level with at least c vertices and let $w = v - \log c$. The total number of impure nodes γ in T is:

$$\begin{aligned}
\gamma &= \sum_{i=1}^{v} m^{i-1} + c(u - v + 1) \\
&= \frac{(1 - m^v)}{1 - m} + c(\log n - (w - \log c) + 1) \\
&= \frac{1}{1 - m} + c(\log \frac{n}{c} - w + 1 - \frac{m^w}{1 - m}) \\
&\leq \frac{1}{1 - m} + c(\log \frac{n}{c} + \frac{m}{m - 1})
\end{aligned}$$

since $0 \leq w < 1$ and $f(w) = -w - \frac{m^w}{1-m}$ is a convex function. For $0 \leq w \leq 1$, $f(w)$ is maximized at $w = 0, 1$, where $f(0) = f(1) = \frac{1}{m-1}$. Therefore there are at most $\gamma - c = \frac{1}{1-m} + c(\log \frac{n}{c} + \frac{1}{m-1})$ impure internal nodes in T. Each internal node has exactly m children, therefore T has at most $\frac{m}{1-m} + mc(\log \frac{n}{c} + \frac{1}{m-1})$ nodes.

The following theorem computes the upper bound of the total cost of the identification scheme when n is a power of m (equation 1).

Theorem 3. *Let n be a power of $m > 1$. Then for c corrupted nodes in the n nodes, $1 \leq c \leq n/m$, the total cost $G(m, n, c)$ of the identification algorithm is upper bounded by $\sum_{i=1}^{u} H[i]$ where H is a sequence of size $u = \lceil \log_m n \rceil$, defined as:*

$$H[i] = \begin{cases} m^{i-1}(\log \frac{n}{m^{i-1}} + 1) \text{ if } i < v \\ mc\ (\log \frac{n}{m^{i-1}} + 1), \text{ if } i \geq v \end{cases} \tag{4}$$

where $v = \lceil \log_m c \rceil$ and log *denotes* $\log_2$.

Proof. Let tree T be the m-ary identification tree whose root vertex is associated with a set of size n. Let sequence element $H[i]$ represent the total cost of the identification algorithm in level i of the identification tree T. Each level i of T has m^{i-1} vertices, where at most c are impure. Also each vertex at level i has exactly $\frac{n}{m^{i-1}}$ nodes. Level v of T is the first level where T has at least c vertices. Therefore at level $i < v$, all m^{i-1} vertices are impure. Since each test has a cost of at most $(\log p + m)$, where p is the number of nodes tested, the total cost of

each level $i < v$ is upper bounded by $m^{i-1}(\log \frac{n}{m^{i-1}} + m)$. Now consider level $i \geq v$. Then each level has at most mc impure nodes of size m^{i-1}. Therefore total cost of each level $i \geq v$ is upper bounded by $mc(\log \frac{n}{m^{i-1}} + m)$.

Upper bound when n is not a power of m In the general case when n is not a power of m, we cannot use the approach of [3] (which they solved for $m = 2$) as the number of possible ways the corrupted nodes are distributed within each subtree explodes (analogous to the combinatorial, ball in the bucket problem). Instead we propose a novel model, inspired by the work of Fiat and Tassa [5] in the context of dynamic traitor tracing (DTT)[4]. We introduce the notion of a *path trace*, defined with respect to a particular corrupted node. The path trace traces the identification path of that node in the identification tree T. Informally we say a path trace D for corrupted node u is rooted at the vertex v in tree T that the identification algorithm separates it from the other corrupted nodes in the network. The trace includes all the vertices in the path between v and the leaf vertex associated with set $\{u\}$. Therefore each time an impure vertex v' in T has more than one impure child, then the algorithm learns that the node set at v' contained more than one corrupted node, and thus a new tree trace D' is generated. Figure 2 shows the paths generated for an example identification tree.

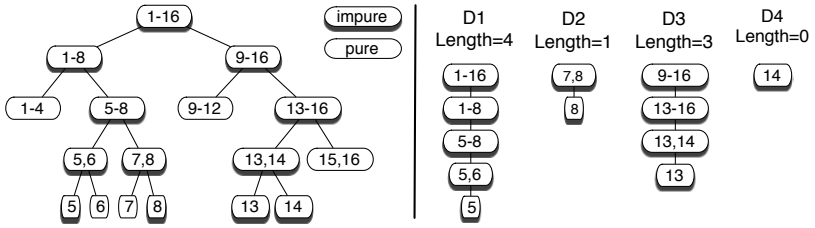


Fig. 2. Path traces for corrupted nodes 5,8,13 and 14.

Note that although a path trace is not unique to a given node, the set of path traces generated is unique. We can therefore determine the set of path traces in a network without associating them to a particular corrupted node.

Claim. A path trace of length ℓ generates $m\ell$ partitions where there are m partitions of sizes $\{m^{\ell-i}\}$, $i = 1, \cdots, \ell$.

Proof. The path trace is a path on an m-ary tree and each internal vertex on the path has $(m-1)$ other siblings that are also tested. Also a path trace of length ℓ has a root vertex associated with m^ℓ nodes. Thus at level i of the path, the vertex is associated with $m^{\ell-i}$ nodes.

[4] The DTT model is different from our model as in each round in DTT, only one receiver node misbehaves.

Consider an identification tree T generated by algorithm 1, for a network of n nodes.

Claim. Let $n = m^h$ where $h \in \mathbb{Z}^+$. Then define sequence P as:

$$P = \{h, \{h-1\}^{m-1}, \{h-2\}^{m(m-1)}, \{h-3\}^{m^2(m-1)}, \cdots\} \tag{5}$$

where $\{y\}^x$ denotes the value y repeated x times. The first c elements in P represent the tight upper bound on the length of the path traces generated when n contains c corrupted nodes.

Proof. Since n contains at least one corrupted node, the first path trace D_1 is rooted at the root of T and thus has length h. A new path trace is generated any time a vertex in T contains more than one impure child. We want to maximize the lengths of the path traces. Therefore on level 2 of T, up to $(m-1)$ path traces can be generated of length $(h-1)$; at level 3, up to $m(m-1)$ path traces can be generated of length $(h-2)$, and so on. In general, in level i, up to $m^i(m-1)$ path traces of length $(\log_m n - (i-1))$ can be generated. Since one path trace is associated with each corrupted node and there are c corrupted nodes, the set of lengths associated with the generated path traces can be represented by the first c elements of P.

Theorem 4. *Let $m^{h-1} < n < m^h$ where $h \in \mathbb{Z}^+$. Then define sequence P' as:*

$$P'[i] = \begin{cases} P[i] & \text{if} \quad i < x \\ P[i] - 1 & \text{if} \quad (i > x \;\&\; P[i] > 0) \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where P is the sequence defined in equation 5 and the index x is defined as,

$$x = \left\lceil \frac{n - m^{h-1}}{m-1} \right\rceil \tag{7}$$

The first c elements in P' represent the tight upper bound on the length of the path traces generated when n contains c corrupted nodes.

Proof. It is trivial to show that the number of internal nodes x at level h are defined as in equation 7. Thus there are $(m^{h-1} - x)$ leaves in level $(h-1)$ and $(n - m^{h-1} + x)$ leaves in level h.

To find the upper bound of the lengths of the path traces, $\min(x, c)$ of the path traces are associated with a corrupted node at level $(h+1)$ and thus they correspond to a identification tree for m^h nodes. The path traces corresponding to the remaining corrupted nodes have leaves at level h and correspond to a identification tree for m^{h-1} nodes.

We can use theorem 4 to derive the tight upper bound on the total cost of the identification algorithm. Consider identification tree T generated by algorithm 2, for a network of n nodes. Let T contain α complete m-ary trees and one

incomplete m-ary tree, with respective depths $d_1, \cdots, d_{\alpha+1}$. Let $P_1, \cdots, P_{\alpha+1}$ correspond to the set of potential path traces for each of the $(\alpha+1)$ respective subtrees using claims 4.2 and 4. Then let sequence P be composed of the non-increasing ordered set of the path traces $\{P_i, \cdots, P_{\alpha+1}\}$, i.e. $P = \{h, (h-1)^{a_1}, (h-2)^{a_2}, \cdots\}$, where $a_1, a_2, \cdots$ are dependent on the size of the subtrees. If n contains c corrupted nodes, then the length of the generated path traces are bounded by the first c elements in P. We can use claim 4.2 and sequence P to compute the size and number of the partitions that algorithm 2 generates in the worst case and derive total cost by summing the cost of the c path traces.

Finally we derive a closed form expression for the *loose* upper bound on the total cost of the network. This is purely for the purposes of comparison of our work with existing solutions.

Theorem 5. *For c corrupted nodes in a network of size n, the identification algorithm 1 has a communication link cost of $O(c^2 \log^3 n)$.*

Proof. Consider the identification tree corresponding to the identification algorithm, which is of height $\log n$ (theorem 3.1). Then for the worst distribution of corrupted nodes in the network, there exists at most mc partitions in each level. Since partition cost is $\mathrm{O}(\log p)$ for a partition of size p, then the total cost can be loosely bounded by:

$$\begin{aligned} mc \sum_{i=0}^{\log n} \log \frac{n}{m^i} &= mc \left(\sum_{i=0}^{\log n} \log n - \sum_{i=0}^{\log n} \log m^i \right) \\ &= mc \left(\log^2 n - \log m \sum_{i=0}^{\log n} i \right) \\ &= mc(\log^2 n - \frac{\log m \log n}{2}(\log n + 1) = O(c \log^2 n) \end{aligned}$$

Since according to theorem 4.1 and claim 4.2, partition cost of n nodes is $\mathrm{O}(c \log n)$, then total cost is bounded by $\mathrm{O}(c^2 \log^3 n)$.

4.3 An Example

To gain a better intuition of the results and the interaction of partition and test cost as part of total cost, we compute the cost associated with handling an adversary attack in a network of 4096 nodes and analyze the graceful degradation of the network service. For test cost, we use the cost derived by [2]. Although [6] improves on the cost bound (from $\mathrm{O}(\log^2 n)$ to $\mathrm{O}(\log n)$), they do not determine a fixed cost characteristic. Consider an adversary that corrupts an arbitrary number c of corrupted nodes.

Figures 3(a) and 3(b) graph the maximum partition cost and total cost of the m-ary identification scheme, when $m = 2, 4, 8, 64$. The baseline scheme is used

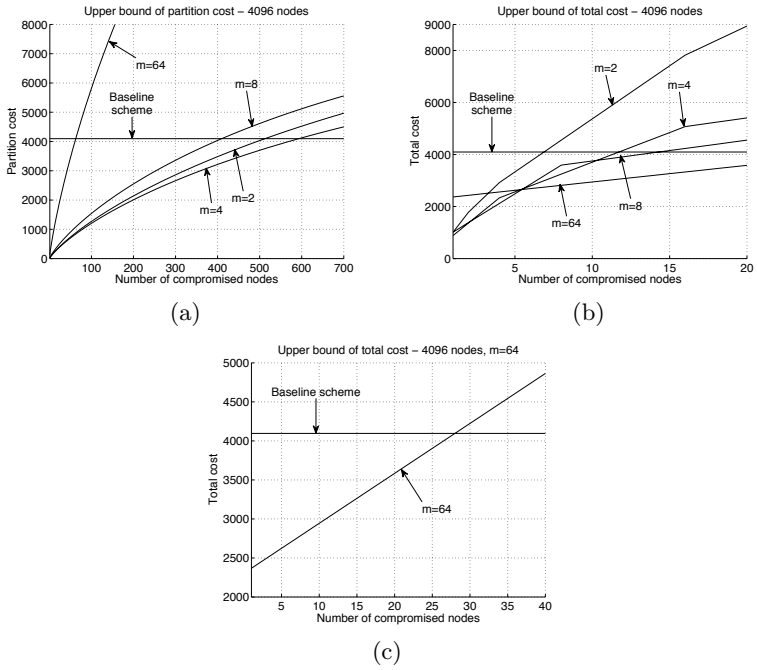


Fig. 3. (a) Partition cost and (b-c) total cost of identification.

in both graphs as a lower bound for when the proposed identification scheme is effective and efficient. Figure 3(a) verifies the intuition that for a fixed number of corrupted nodes in the network c, the number of generated partitions increases with the partition degree m. This increase plateaus when the the identification scheme needs to test every single vertex on the identification tree. In this network configuration, best partition cost (i.e. minimum cost) is achieved when $m = 4$.

The performance of the m-ary identification scheme is best shown in figure 3(b) where the link cost for different values of m is compared with the baseline. As partition degree m becomes larger, the scheme performs better than the baseline scheme for larger number of corrupted nodes, i.e., 7 (for $m = 2$), 12 (for $m = 4$), 14 (for $m = 8$) and 28 (for $m = 64$). However, when $c < 6$ smaller partition degrees perform better. Therefore to optimize total cost (equation 3) a network administrator choose an appropriate partition degree depending on probability of attack, vulnerability of the network as well as the necessary rapidity of the response (since response time is $\mathrm{O}(\log_m n)$).

The effect of partition cost on the total cost is also evident in figure 3(b). In particular the best total cost when $c > 6$ is obtained when $m = 64$. It is thus clear that the test cost is the dominant term in the total cost of the scheme. This is a promising result as test cost is only dependent on the cost of the aggregation-

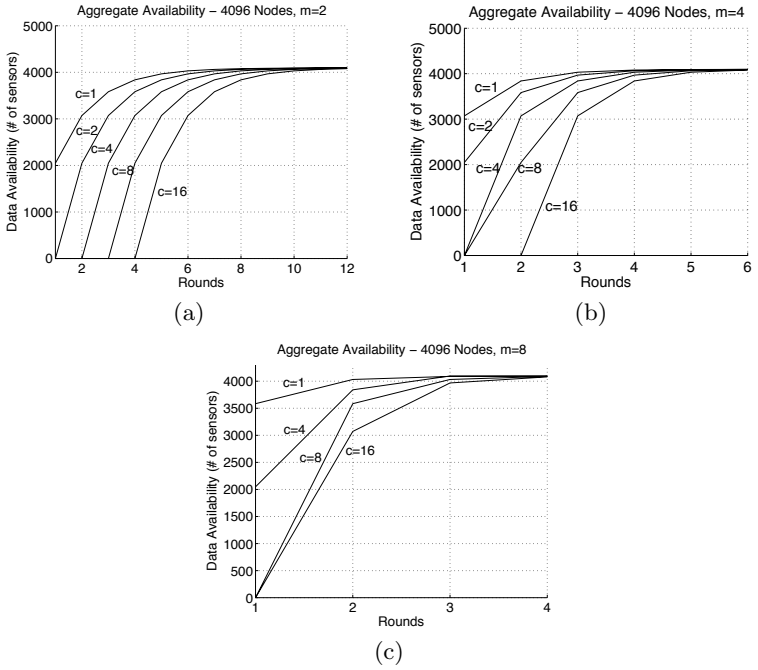


Fig. 4. Data availability at the BS over successive rounds.

verification primitive. More efficient primitives yield better results. In contrast the fixed cost associated with the scheme, the partition cost, is relatively small.

Figure 4 shows the rate of improvement of the network service over the course of the protocol execution. Data is normalized by simply looking at the number of nodes that contribute to the network aggregate in a particular round. The maximum available data for a network of size n with c corrupted nodes, is $n - c$. Although most of the figures follow intuition, they are interesting because they show the rate of improvement of the data availability in the network and how the availability changes across different partition degrees. Fixing the number of corrupted nodes c, as m is reduced, data becomes available in later rounds. For example, for 8 corrupted nodes, the first available data occurs in rounds 3 (for $m = 2$), 2 (for $m = 4$) and 1 (for $m = 8$). This is because in the worst case, for example when $m = 4$, all four partitions generated in the first round are impure. The second round is the first round which contains more partitions than c and therefore the BS is guaranteed to obtain correct data.

4.4 Lower Bound and Average of Partition Cost

Up to now, we have focused on the upper bound or worst case partition cost. To gain an understanding for the behavior of the algorithm in practice, it is

important to also analyze the lower bound and average behavior of the partition cost.

Average. Computing the average partition cost when the number of malicious nodes in the network is fixed leads to a state space size exponential in the number of nodes in the network. To see this, observe that the number of possible states in each round is equivalent to the classic combinatorial balls-in-the-buckets problem where the balls and the buckets correspond to the nodes and partitions in the network respectively; furthermore there are two types of balls (good nodes and bad nodes). Then the total number of possible states can be computed by the product of the number of states in each round. This leads to a combinatorial explosion in the number of states and therefore is hard to compute exactly.

Lower Bound. The lower bound of the partition cost can be obtained by minimizing the number of vertices in the identification tree T. According to the algorithm, every internal vertex of T must be associated with an impure set and there must exist exactly c impure leaves. Partition cost can be minimized when both the number of pure leaves and the internal vertices in T are minimized. Figure 5 presents two different distributions of 3 corrupted nodes in the network which yield maximum and minimum partition cost.

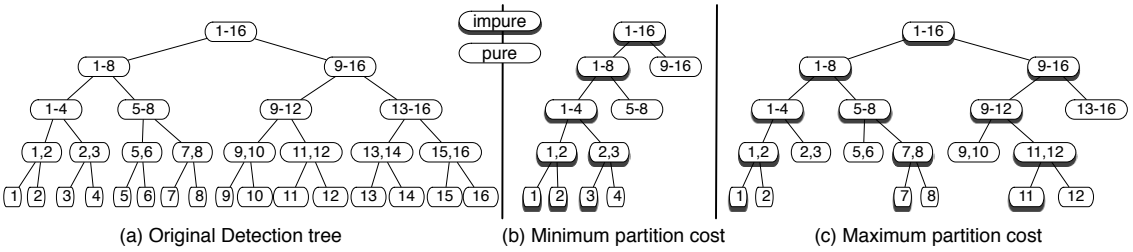
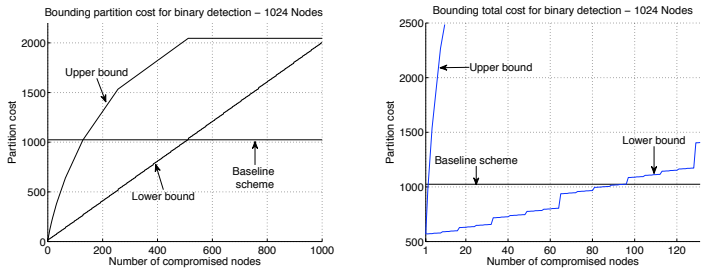


Fig. 5. Partition cost depends on how the corrupted nodes are distributed in the network.

Figure 6(a) and 6(b) respectively graph the lower and upper bounds of the partition cost and the total cost of the identification algorithm for a network of 1024 nodes. In the best case, the performance of the identification algorithm is better than the baseline even up to 96 corrupted nodes. However in the worst case, the scheme performs worst than the baseline scheme when simply 3 nodes are corrupted.

5 Conclusion and Future Work

Adversary attacks against data aggregation in ad hoc networks can have disastrous results, whereby a single corrupted node can affect the perceived measure-



(a) (b)

Fig. 6. Bounding (a) partition cost and (b) total cost for binary identification algorithm.

ments of large portions of the network by the base station. Current approaches to handling such attacks either aim exclusively at the detection of attack [2, **?**] or provide inefficient ways of identifying corrupted nodes in the network, with respect to the baseline scheme. In this work, we presented a group-based approach to handling adversary attacks in aggregation applications. The proposed scheme provides an efficient method in identifying corrupted nodes while ensuring continuous, but gracefully degraded service during the attack period. Our analysis results in a precise cost-base characterization of when in-network aggregation retains its assumed benefits in a sensor network operating under persistent attacks. Our scheme is most effective when the adversary has corrupted a small fraction of the nodes in the network.

Although our work provides promising results in divide-and-conquer handling of attacks in aggregation applications, we have assumed a simplified adversary model. In the following we identify several directions to improve our scheme.

Intermittent Adversary. An important assumption in our analysis of the proposed framework is that the adversary persistently misbehaves. Although this is a promising first step, in practice an adversary can be adaptive and can change its behavior to avoid detection. In particular, instead of always misbehaving, the adversary can choose to misbehave *intermittently* and 'hide' in certain rounds. Although our algorithm can handle an intermittent adversary in most cases, a corrupted node can escape detection by hiding in special cases (see appendix C). We plan to extend our scheme to handle intermittent adversaries.

Localized Adversary. We have clearly shown that the cost of the identification scheme is directly related to the distribution of the corrupted nodes on the aggregation tree. identification cost increases as nodes are more uniformly distributed along the aggregation tree (*viz.* figure 5), and similarly on the physical routing tree ([6] show that the aggregation tree is built so that physically neighboring nodes are also neighbors on the aggregation tree). Alternatively, cost can be minimized when corrupted nodes are distributed so that they form a contiguous cluster. This is a promising observation as in practice an adversary is localized – compromising uniformly in the network requires not only a greater

deal of network access, but also increased risk of physical detection. We can therefore expect the cost of identification to be closer to the lower bound (best case) than the upper bound (worst case). We plan on providing more concrete analysis and simulations to confirm our expectation.

Detection Error. Many networks can tolerate detection error if the said error is insignificant, yet can dramatically improve the cost of detection. In our problem, a detection error occurs when a good node is mislabeled by the testing algorithm (a false positive). It is interesting to investigate the effect and best approach of introducing false positives into the system.

References

1. B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
2. H. Chan, A. Perrig, and D. Song. Secure hierarchical in-network aggregation in sensor networks. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 278–287, 2006.
3. D.-Z. Du and F. K. Hwang. Competitive group testing. *Discrete Applied Mathematics*, 45(3):221–232, 1993.
4. D.-Z. Du and F. K. Hwang. *Combinatorial Group Testing and Its Applications*. On Applied Mathematics, Volume 12. World Scientific, second edition, 2000.
5. A. Fiat and T. Tassa. Dynamic traitor tracing. In *Advances in Cryptology - CRYPTO'99*, 1999.
6. K. Frikken and J. A. Dougherty IV. Efficient integrity-preserving scheme for hierarchical sensor aggregation. In *Proceedings of the 1st ACM conference on Wireless Network Security (WiSec)*, 2008.
7. P. Haghani, P. Papadimitratos, M. Poturalski, K. Aberer, and J.-P. Hubaux. Efficient and robust secure aggregation for sensor networks. In *3rd IEEE Workshop on Secure Network Protocols*, pages 1 – 6, Oct 2007.
8. L. Hu and D. Evans. Secure aggregation for wireless networks. In *Workshop on Security and Assurance in Ad Hoc Networks*, pages 384–392, 2003.
9. B. Przydatek, D. Song, and A. Perrig. SIA: Secure information aggregation in sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, 2003.
10. J. Staddon, D. Balfanz, and G. Durfee. Efficient tracing of failed nodes in sensor networks. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pages 122–130, 2002.
11. D. Wagner. Resilient aggregation in sensor networks. In *Proceedings of the 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks*, 2004.
12. Y. Yang, X. Wang, S. Zhu, and G. Cao. SDAP: A secure hop-by-hop data aggregation protocol for sensor networks. In *ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 356–367, 2006.

A Group SHIA

For the sake of brevity, we only describe the differences between SHIA and GSHIA and we refer the reader to [2] for details and analysis of SHIA. GSHIA has four main phases: query dissemination, grouping, aggregation-commit and result checking.

Query Dissemination. The request message the BS broadcasts also includes the following grouping information: the size of the new partitions as well as the ID of the groups that were found impure in the previous round.

Aggregation-Commit. Nodes within a group compute a cryptographic commitment structure over their data values and the aggregation process as in SHIA. Also to allow the resolution of the group memberships, each leaf vertex also computes a Bloom filter that probabilistically summarizes the node membership set. The filter is then forwarded to the parent internal vertex, who aggregates the filters using the bit-wise OR function. Additionally each leaf vertex computes an authentication tag over a fixed message and forwards this to its parent node. The tags are aggregated using the bit-wise XOR function to form the *group tag*. The group filter and tag are used by the BS to determine the group membership sets.

Grouping. In this phase, node grouping is conducted through the selection of leader nodes for each group. This phase is executed in parallel with the aggregation-commit phase. Whenever a node performs the aggregation and commitment operations, it also determines if it is a group leader by comparing the group size with the target size broadcast by BS in the request message. If a node is selected as group leader, then all the nodes in its subtree which do not as yet belong to a group become its group members. The group leader then computes a message authentication code (MAC) and forwards the group aggregates and commitments along with the MAC to the BS without further aggregation along the way. If the node is not selected a group leader, it simply forwards its aggregation and commitment values to its parent node, where they are aggregated further. In this way, groups are iteratively generated starting from the leaf nodes and approaching the BS or the root of the network spanning tree. At the end of the aggregation-commit process, all remaining nodes which do not belong to a group are grouped together with the BS acting as their group leader.

Result Checking. At the end of the aggregation-commit and grouping phases, each group leader has reported their aggregation results and commitment values to the base station. The base station first determines the group size and the membership set of each group. This is done by narrowing down the potential membership sets of a group based on the location of the group leader, the group size and the group Bloom filter. The correct membership set can be verified by the aggregated group tag. Once the membership set of a group has been determined, the group size can also be verified. The BS then authenticates the final commitment values of each group and disseminated them to the respective groups.

The result checking is the distributed verification process as in SHIA, where each group verification code is forwarded to the BS by the group leader. When the BS receives all the group confirmation codes, it accepts the group aggregates if it can verify that all group members have individually verified the correctness of the aggregation protocol. The BS discards any group aggregate which is not verified by all group members and classifies the associated group as suspicious. The final network aggregate is computed over all groups which are deemed correct.

Communication Complexity. The aggregation verification process has a link congestion of $O(\log n)$ where n is the size of the group [6]. GSHIA introduces the following additional messages in the query dissemination and aggregation-commit phases of the scheme: (1) group IDs, (2) Bloom filter output and (3) group tag. Messages (2) and (3) are fixed size and message (1) depends on the number of active partitions in the network, m. Thus the final link congestion for GSHIA is $O(\log n + m)$.

Security Analysis of Grouping. We must show that a malicious node cannot force the BS accept a false grouping. A group membership set is summarized via the Bloom filter. The BS can verify that the filter output has not been changed by verifying the validity of the group tag, which is the XOR of the MACs of the group members. Assuming that the MAC scheme is secure, then a malicious node can at best forge the group tag with negligible probability. Hence if the BS can verify the group tag associated with the Bloom filter and ensuring that each node in the network belongs to at most one group, it knows with overwhelming probability that the grouping is correct.

B Group Testing Results for $m = 2$

Du and Hwang [3] presented a bisecting algorithm, where at each step, if an impure set X is discovered, then the algorithm bisects X and tests the resulting two subsets X_1 and X_2. They propose the following partitioning rule:

Algorithm 4 ***Improved Bisecting Rule***
Input: *Set* X.
Output: *Result sets* X_1, X_2, *such that* $X_1 \cup X_2 = X$.
Bisect X *into subsets* X_1 *and* X_2, *where* X_1 *contains* $2^{\lceil \log |X| \rceil - 1}$ *nodes and* $X_2 = X \setminus X_1$.

This rule partitions the network such that the identification tree is made up of one complete and one incomplete binary subtree. Figure 1 uses this rule to partition the network of 12 nodes. We adapt the following theorem, proved by Du and Hwang [3] for our problem. We refer the reader to the original paper for the proof of the theorem.

Theorem 6. *Given an input set of nodes N, partition degree 2 and the bisecting rule defined by algorithm 4, identification algorithm 1 produces at most* $2c(\log_2 \frac{|N|}{c} + 1) + 1$ *partitions before outputting all the corrupted nodes.*

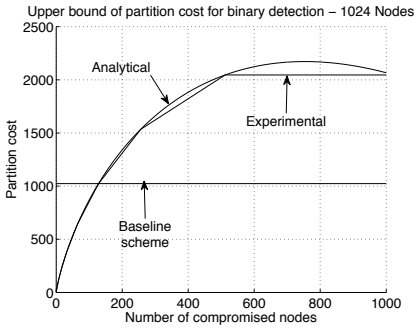


Fig. 7. Comparison of the analytical and numerical bounds with the baseline scheme.

Figure 7 shows the behavior of the partition cost of the binary identification algorithm when the Du and Hwang's partition rule is used and compares this cost with the

total cost of the baseline scheme. The figure shows that when considering the partition cost of the identification algorithm alone, the identification algorithm performs better than the trivial solution for up to 128 corrupted nodes. Therefore if an adversary corrupts more than 128 nodes in the network (or $\frac{1}{8}$th of the total nodes in the network) then aggregation is not a useful primitive for a secure and defensive network.

The figure also compares the analytical partition cost obtained from theorem 6 and the partition cost computed using the tight bounds in theorem 4, which yields exact results. Although the analytical curve follows the numerical curve faithfully, it does produce significant error for $c > 128$, where c denotes the number of corrupted nodes. The experimental curve reaches a plateau at $c = 128$. This can be explained as if $c = 128$, the worst distribution of corrupted nodes requires the testing of every single node; consequently no extra test is required for $c > 128$.

C Intermittent Adversary Analysis

In the following, we analyze the behavior of the proposed identification scheme for an intermittent adversary first in a setting where the adversary corrupts a single node in the network and then when it corrupts $c > 1$ nodes. For simplicity we assume binary identification (i.e. $m = 2$), though our analysis can be trivially extended to the m-ary case. Consider a corrupted node u_1 in set $S = \{u_1, \cdots, u_n\}$ that misbehaves. Since S tests impure, in the following round τ, the identification algorithm bisects S into sets S_1, S_2 and tests the purity of each set. Without loss of generality, assume $u_1 \in S_1$.

When only a single node is corrupted.
If u_1 misbehaves in round τ, S_1 test impure and S_2 tests pure. The algorithm traces the location of the malicious node to S_1 and repartitions S_1 in round $(\tau + 1)$. Otherwise if u_1 behaves correctly, both S_1, S_2 test pure. Since the detector knows there must exist at least one bad node in S (as it tested impure), it assumes correctly that the adversary is 'hiding'. The detector can then choose to keep the suspicious sets S_1, S_2 isolated until the hidden corrupted node misbehaves and the algorithm can further trace it.
Detector Strategy: The detector follows algorithm 1 except that it advances to the next round only when malicious behavior is detected; otherwise the detector repeats the same round.

When $c > 1$ nodes are corrupted.
Assume nodes $u_1, \cdots, u_c \in S$ are all corrupted. We consider two cases in round τ, when nodes $u_1, \cdots, u_c$ all choose to behave correctly, or when at least one of $u_1, \cdots, u_c$ misbehaves. In the former case, since the detector knows that there must exist at least one bad node in S, it assumes the adversary is hiding and thus keeps the suspicious sets isolated and waits for the hidden node(s) to misbehave again (this reduces the strategy to the single corrupted node setting). In the second case when at least one of $u_1, \cdots, u_c$ misbehaves in round τ, then the identification trace follows the misbehaving nodes. Assume $u_1 \in S_1$. If S_1 tests impure, then S_1 remains a suspicious set in round $(\tau + 1)$ and u_1 can still be detected in future rounds. However if S_1 tests pure (and S_2 tests impure), then u_1 escapes identification. This is because all the detector knows is that there is at least one corrupted node in S and subset S_2 has tested impure; it cannot make any other conclusions about the existence of any hidden nodes.

Therefore, an intermittent corrupted node u can escape identification by hiding in round τ if set S_1 which contains u does not have any other misbehaving nodes and at least one other set $S_i \subset S \setminus S_1$ tests impure.