Resilient Data Collection in Smart Grid

Tianyuan Liu[1], King-Shan Lui[2], Haiming Jin[1], and Klara Nahrstedt[1]

[1]Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA
[2]Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong

***Abstract*—Sensors and measurement devices are widely deployed in Smart Grid (SG) to monitor the health of the system. However, these devices are subject to damage and attack so that they cannot deliver sensing data to the control center. In tree-based data collection schemes, a relay failure can further lead to unresponsiveness of all the devices in its sub-tree. In this paper, we study the resiliency issue in collecting data from SG measurement devices. We first design a protocol that guarantees successful data collection from all non-faulty devices in a backup-enabled tree structure. Then, we formulate the tree construction problem to optimize data collection time. Since the formulated problem is NP-hard, we propose a heuristic algorithm to solve it. We evaluate our algorithm using a real utility network topology. The experiment results show that our algorithm performs well in large scale networks.**

I. Introduction

To monitor the health of the Smart Grid (SG), massive number of sensors or measurement devices will be installed to collect various real-time data. These data have to be collected efficiently and accurately to facilitate robust monitoring and control. Nevetheless, as the sensors are deployed in an open area, they may be subject to different forms of attack and damage, and become faulty after installation. The data collection infrastructure thus should be secure and resilient that data should still be reported in a timely fashion when some sensors, which also act as data relays, fail. In this paper, we describe our resilient data collection protocol that allows sensors to reconstruct their data reporting paths when some nodes fail.

We consider the scenario where a large number of measurement devices (MDs) are deployed in a large area. A mobile data collector (DC), which is a device that has abundant memory, is responsible to collect the data. The DC will follow a certain fixed data collection path. When an MD falls in the communication range with the DC, it can talk to the DC directly. To save cost, the DC would not visit every MD. Therefore, MDs that are not visited have to send their data first to other MDs, and these MDs deliver the data to the DC in a hop-by-hop manner. The paths that the data go through would form multiple trees rooted at the MDs that can be directly connected to the DC. Figures 1 and 2, where thickened lines represent the tree links, illustrate two possible data collection structures with different properties with the same network. Only MD_1, MD_2, and MD_3 can talk to the DC directly when it passes through. They are all potential tree roots. However, it is not necessary for all of them to be roots. In Figure 2, MD_1 is a root but MD_2 is not. Each MD is connected to at most one root, and the MDs form multiple data collection trees.

We assume the same tree collection structure would be used for multiple data collection instances. The frequency of data collection would depend on the application and the accuracy needed [1], [2]. Before the first data collection, the tree collection structure is computed by the system adminstrator (SA) based on the topology and the DC data collection path. The SA then informs the MDs the tree structure. Each MD then knows whether it has to serve as the root, and which nodes are its parent and children on the tree. As the DC can only talk to an MD when it moves to the proximity, we assume the DC talks to the roots one by one. We further assume that data is collected on demand, and thus a root node knows that data have to be collected only when it is notified by the DC. The time to collect data on a certain tree is thus related to the height of the tree. The amount of time needed for the whole data collection process is the sum of total height of all trees. We will consider this performance metric in our mechanism. We will also consider the *key leakage probability* in the tree construction, which will be described in further details in Section III.

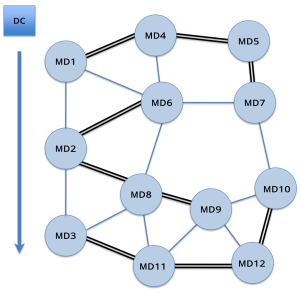

Fig. 1. Tree structure with three roots.

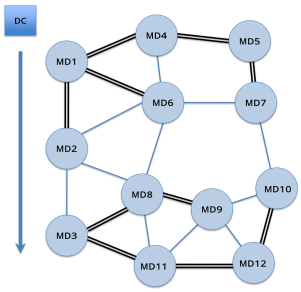

Fig. 2. Tree structure with two roots.

Since the MDs are exposed to an insecure environment, an MD may fail and stop working. When this happens, the faulty MD can no longer help other MDs to relay their data, and the DC cannot collect the information needed. To avoid this from happening, each MD is associated with a "backup parent" such that when the parent MD in the original tree structure fails,

the backup parent will serve as the relay to send data to the DC. The selection of backup parents should also minimize the height of trees while preserving connectivity. Specifically, the backup links should not introduce cycles into the data collection trees. A previous work [3] suggests selecting grandparent, uncle and cousin nodes as backup parents on the same tree. However, some nodes, e.g. the children nodes of a tree root, do not have a grandparent/uncle/cousin such that they cannot find proper backup parents. Due to the multi-tree structure, in this work we consider an alternative to select backup parents for a node from its neighbor trees. We formulate the problem as *Resilient Data Collection Tree Problem (RDCTP)* and develop *Two-step Greedy Heuristic* to solve it. We perform simulation on real world power grid dataset.

II. Related Work

Data collection in sensor networks by a mobile sink has been actively studied [4]. Most work aim at optimizing energy usage while reducing data reporting latency. Due to the massive number of sensors, a hierarchical data collection structure is usually adopted. Cluster heads are selected to collect data from sensors within its neighborhood. Cluster heads then report the data to the sink/data collector. Some recent studies on how to select cluster heads to balance energy and latency can be found in [5], [6]. As energy harvesting has been proposed to prolong the lifetime of a sensor, some researchers study data collection with this emerging technology [7], [8]. To reduce traffic, compression techniques are studied to improve the data collection performance [9], [10]. To the best of our knowledge, there is no representive study on resilient data collection in sensor networks that can be applied in our data collection scenario.

Data collection in smart grids has been studied in the security and data aggregation aspects. Nevertheless, there are not many studies on resilience identified. [11] studies where to put extra relay nodes to provide fault tolerance in the overhead transmission lines. As the topology of the transmission grid is linear, the mechanism cannot be applied in the tree structure. [12] considers the robustness of data collection in advanced metering infrastructure. Similar to our work, data are collected through a tree structure. A primary tree is first built and then backup links are identified to provide resilience. The proposed algorithm aims at finding the minimum set of links to form the resilience tree. The authors in [3] study the performance of different backup parent selection mechanisms. Our work differ from these works in two aspects: first, [3], [12] focus on backup selection in single tree structure and assume that tree root does not fail. However in our problem, we aim at selecting backup parents among multiple trees and we assume tree roots can be faulty. Second, besides connectivity issue, we also consider other objectives and constraints in backup mechanisms such as security and latency.

III. System Model

A. Security Model

Before we describe our resilient tree construction protocol, we first explain the security requirement in data collection. The data reported by each MD should be protected in both integrity and privacy. The data should be read by the SA only, but not intermediate MDs which help relaying the data. On the other hand, to save bandwidth, it would be desirable for intermediate MDs to verify whether the message has been tampered and should be dropped. That is, intermediate MDs should be able to authenticate the message but not read the data carried in the message.

Similar to our earlier work [13], we developed the secure tree-based data collection mechanism that allows MDs to report private data to SA while facilitating intermediate MDs to verify the messages. *Diffie-Hellman* key exchange is used to establish an encryption key for the data. Integrity check along the tree path is supported by a *group key* which is known by all the members in the tree. If an adversary knows the group key, he can forge messages without being detected by integrity check. Although the SA can finally detect the data carried in the message are not legitimate, network resource will be wasted in transmitting the message. As the MDs may not be in a very secure environment, there is a risk that the group key is leaked from a group member. We model the leakage probability of MD_i to be $p_{\text{leak}}(i)$, and control the risk of group key leakage under a security threshold.

B. Resiliency Model

Apart from security attacks, MDs are subject to physical damages that they may fail and stop functioning. When an MD fails and stops, it can no longer report data. It also cannot relay information for other MDs. More precisely, a failed MD would lead to the loss of the data of its whole subtree, which may be very serious. To avoid this from happening, the data collection structure has to be resilient that when a certain node fails, the loss of data should be minimal. Our resilience strategy is to identify a "backup parent" for each node. When the original parent no longer works, a node would forward the data to the backup parent so that the data can still reach the DC.

We consider the *fail-stop model* for all MDs, i.e., once an MD fails, it may not recover until after several data collection rounds. We assume that all MDs are collecting the same types of data, and that MDs (including root) can fail with a probability p_{fail} identically and independently. Furthermore, we consider at most one failure for each data collection round and assume that failures are detected before a data collection round happens, i.e., the backup links should have been turned on when a data collection command arrives.

IV. Protocol Design

A. Protocol Overview

Our protocol can be described in four sub-routines: *tree computation*, *key distribution*, *secure data collection* and *backup switch*.

The tree computation is done at SA. The detail of how trees are constructed in this sub-routine will be discussed in Section V. In our previous work [13], we designed secure protocols to distribute group keys and to collect data along tree branches. We first briefly introduce the key distribution and secure data collection protocol. Then, we describe the backup switch protocol in detail. The detail of key distribution and data collection protocol can be found in [13].

B. Key Distribution

After tree structure is computed at SA, it distributes two types of keys to all the MDs before any data collection round. Specifically, each MD gets a symmetric key sk and a group key gk. The symmetric key is shared between the MD and SA to encrypt and decrypt the data. Diffie-Hellman key exchange is used to establish this key. The group key is shared among all the members in a tree. This key is initiated by SA and sent to every group member.

As keys are distributed, we also notify MDs about their parents and children information. Such information for a non-root MD_i contains the identities and public keys of following instances: (1) A primary parent $P(i)$; (2) A backup parent $BP(i)$; (3) A list of primary children $CH(i)$; (4) A list of backup children $BCH(i)$. When no failure is detected, the tree structure is described by the primary parents and children.

C. Secure Data Collection

Suppose we have a data collection chain on a tree branch $\mathrm{DC} \leftarrow \mathrm{MD}_0 \leftarrow \cdots \leftarrow \mathrm{MD}_k$, where MD_0 is the root of this tree. The secure data collection protocol operates as follows:

a) Command Forward: As MD_0 receives a data collection command from DC, it first looks up the its primary children $CH(0)$ from memory, and forwards the data collection command to all its children. After that, it prepares the data by encrypting its raw data by the symmetric key, i.e., $DATA_0 \leftarrow \mathsf{Enc}(RAWDATA_0, \mathsf{sk}_0)$. Meanwhile, it waits for all the children to send data report. Every non-leaf MD follows the same process on the tree branch until the data collection command reaches the tree leaf MD_k.

b) Data Report: To facilitate other MDs to check message integrity without understanding the raw data, MD_k computes the hash of the encrypted data using the Diffie-Hellman half key. The half key is encrypted using the group key. That is, MD_{k-1} can use the group key to retrieve the Diffie-Hellman half key and verify the hash sent by MD_k. After collecting all data from its children, MD_{k-1} sends its report to its parent.

D. Backup Switch

Once parent failure is detected by its child MD, the child MD contacts its backup parent to obtain the group key of the backup tree. It stores two group keys so that it uses the old group key to verify hashes from its children, and uses the new one to create hashes for its backup parent.

We use the example topology shown in Figure 3 to demonstrate the backup switch protocol. In this topology, two trees are constructed with roots MD_1 and MD_5 respectively. MD_1 is the primary parent of MD_2, and MD_5 is its backup parent. To distinguish the group keys used in different trees, we denote the group key shared by MD_1, MD_2, MD_3, and MD_4 as gk_1 and the group key shared by MD_5, MD_6, and MD_7 as gk_2.

As soon as MD_2 finds out that MD_1 fails, it looks up its backup parent $BP(2) = \mathrm{MD}_5$ from memory and sends a *backup switch request* to MD_5. Upon receiving the request, MD_5 first checks that MD_2 is a legitimate backup child if $\mathrm{MD}_2 \in BCH(5)$. Then, MD_5 marks MD_2 as a primary child and sends the group key gk_2 to MD_2 using public key encryption. At this moment, MD_2 possesses two group keys gk_1 and gk_2.

When a new data collection command arrives at MD_5, it will forward the command to its children including MD_2 since MD_2 is marked as a primary child. MD_2 thus continues to forward this command to MD_3 and MD_4 so that command will reach all the MDs in the sub-tree of MD_1 even though MD_1 has failed. In the data report process, MD_3 and MD_4 still use gk_1. MD_2 can authenticate their messages since it has gk_1. When MD_2 reports data to MD_5, it uses the new group key gk_2. Since MD_5 possesses gk_2, it is able to perform the integrity check.

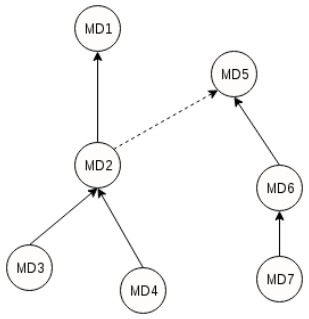

Fig. 3. Backup Topology Example.

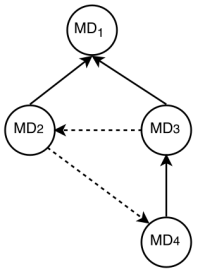

Fig. 4. Cycle Breaks Connectivity.

V. Resilient Data Collection Tree Problem

In this section, we describe how data collection trees are constructed in the tree computation phase. We formulate the problem as Resilient Data Collection Tree Problem (RDCTP) using integer programming, and propose a Two-step Greedy Heuristic.

A. Problem Description

We assume that SA knows the locations of all MDs as well as the route of DC. Hence, a set of MDs along the route can be selected as candidate roots. We denote the topology of MDs as a directed graph $\mathcal{G} = \langle \mathcal{M}, \mathcal{E} \rangle$, where $\mathcal{M}$ is the set of MDs and $\mathcal{E}$ is the set of edges. If MD_i and MD_j can communicate directly without the help of any relay, two directed edges $e_{i,j}$ and $e_{j,i}$ are present in $\mathcal{E}$. In addition, we use $\mathcal{R}$ to denote the set of candidate roots. Since DC does not traverse through the whole graph, the number of candidate roots is much smaller than the number of MDs, i.e., $|\mathcal{R}| << |\mathcal{M}|$.

In order to achieve fast data collection, our objective is to minimize the time to collect data from all the tree roots. The data collection time on a tree depends on the height of the tree. Once the tree structure is determined, the latency can be calculated by the depth-first search (DFS) algorithm.

The resiliency issue is addressed as two constraints in the construction. First, since MDs on a tree share the same group key, the adversary can break the group key by compromising any one of these MDs. We model the possibility that a group key is leaked from MD_i as a probability $p_{\text{leak}}(i)$, and we want to control the probability that the group key is leaked from any tree under a threshold P_{th}.

Second, we model the possibility that an MD fails as a probability p_{fail}. To tolerate primary parent failure, each MD should have a backup data collection path.

The backup paths should not form any cycle with other primary or backup paths. Figure 4 shows a how cycles can affect connectivity of backup paths. The solid lines represents the data collection links of primary parents, and the dashed lines represent those of backup parents. When MD_1 fails, both MD_2 and MD_3 are not connected to any root node.

Previous work [3], [12] address the cycle issue by selecting certain types of nodes as parent. Particularly, three types of backup candidates are identified: (1) Grandparent – parent of parent node; (2) Uncle – sibling of parent node; (3) Cousin – children of uncle node. Unfortunately, this approach does not work in our model since we assume tree roots can be faulty. These three types of nodes do not exist for any child of a tree root since the tree root does not have a parent or sibling. Therefore, instead of selecting backup parents from the same tree, we select backup parents from a different tree.

B. Problem Formulation

Before introducing the detail, we describe the general idea of the formulation. We address the objective of minimizing data collection time from a "path" perspective. Let $\mathcal{P}_{ij}^k$ be the k-th pre-computed shortest path from MD_i to MD_j, $L_{i,j}^k$ be the latency of path $\mathcal{P}_{ij}^k$, and use $\mathcal{K}_{ij}$ as a set of indices of all the paths from MD_i to MD_j. Then, the time to collect data from a sub-tree rooted at MD_j depends on the longest path on the sub-tree

$$D(j) = \max_{i \in \mathcal{M}} \sum_{k \in \mathcal{K}_{ij}} x_{ij}^k L_{ij}^k, \tag{1}$$

where x_{ij}^k is an indicator variable such that

$$x_{ij}^k = \begin{cases} 1, & \text{if } \mathcal{P}_{ij}^k \text{ is selected as a path from } \text{MD}_i \text{ to } \text{MD}_j; \\ 0, & \text{otherwise.} \end{cases}$$

When there is no failure in the MD network, the total time to collect data from all the candidate roots can be expressed as $\sum_{j \in \mathcal{R}} D(j)$.

Now we consider the situation where MD_i's primary parent fails. As described in the previous sections, as MD_i detects the failure of its primary parent, it contacts its backup parent and joins the backup tree. As shown in Figure 3, when MD_2 switches to its backup parent MD_5, it will introduce more latency on the sub-tree rooted at MD_5 if the latency on path $\text{MD}_4 \rightarrow \text{MD}_2 \rightarrow \text{MD}_5$ is larger than that on path $\text{MD}_7 \rightarrow \text{MD}_6 \rightarrow \text{MD}_5$. Following this intuition, we define the "penalty" introduced when MD_i's primary parent fails as:

$$F(i) = \max \left\{ \sum_{q \in \mathcal{N}_i} y_{iq} L_{iq} + D(i) - \sum_{q \in \mathcal{N}_i} y_{iq} D(q),\, 0 \right\} \tag{2}$$

where $\mathcal{N}_i$ is the set of all neighbors of MD_i and y_{iq} is an indicator variable such that

$$y_{iq} = \begin{cases} 1, & \text{if } \text{MD}_q \text{ is selected as backup parent of } \text{MD}_i; \\ 0, & \text{otherwise.} \end{cases}$$

$\sum_{q \in \mathcal{N}_i} y_{iq} L_{iq} + D(i)$ represents the latency introduced by the sub-tree rooted at child MD_i, and $\sum_{q \in \mathcal{N}_i} y_{iq} D(q)$ represents the latency of the sub-tree rooted at backup parent MD_q. Therefore, if the latency introduced by MD_i is larger than the original latency of MD_q, the penalty of switching MD_i to the backup parent will be non-zero.

We consider only single MD failure and suppose each MD fails independently with an identical probability p_{fail}. The total penalty due to MD failure can be expressed as $p_{\text{fail}} \sum_{i \in \mathcal{M}} F(i)$.

As a security constraint, we formulate the key leakage probability of MD_i as $p_{\text{leak}}(i)$. Then, the key leakage probability of a tree $\mathcal{T}$ can be expressed as $P_{\text{leak}}(\mathcal{T}) = 1 - \prod_{i \in \mathcal{T}} (1 - p_{\text{leak}}(i))$.

In order to control the key leakage risk, we require the leakage probability on every tree to be smaller than a predefined threshold $P_{\text{leak}}(\mathcal{T}) \leq P_{\text{th}}$. Furthermore, we assume identical leakage probability at each MD, so that the probability threshold is equivalent to a threshold of the maximum number of nodes on a tree N_{th}. The relationship of P_{th} and N_{th} can be expressed as $P_{\text{leak}}(\mathcal{T}) = 1 - \prod_{i \in \mathcal{T}} (1 - p_{\text{leak}}(i)) = 1 - (1-p)^{|\mathcal{T}|} \leq P_{\text{th}}$. And thus, we have $|\mathcal{T}| \leq \frac{\log(1-P_{\text{th}})}{\log(1-p)} = N_{\text{th}}$.

Now, we formally present the Resilient Data Collection Tree Problem (RDCTP) as follows

$$\min \sum_{j\in\mathcal{R}} D(j) + p_{\text{fail}} \sum_{i\in\mathcal{M}} F(i) \tag{3}$$

$$\text{s.t.} \sum_{j\in\mathcal{R}} \sum_{k\in\mathcal{K}_{ij}} x_{ij}^k = 1, \forall i \in \mathcal{M} \tag{4}$$

$$\sum_{k\in\mathcal{K}_{ij}} x_{ij}^k \le 1, \forall i,j \in \mathcal{M} \tag{5}$$

$$\sum_{q\in\mathcal{N}_i} y_{iq} = 1, \forall i \in \mathcal{M} \tag{6}$$

$$\sum_{k\in\mathcal{K}_{iq}} x_{iq}^k + y_{iq} \le 1, \forall i \in \mathcal{M}, q \in \mathcal{N}_i \tag{7}$$

$$y_{ip} x_{pj}^m + \left(\sum_{k\in\mathcal{K}_{iq}} x_{iq}^k \right) x_{qj}^l \le 1, \forall i \in \mathcal{M}, p,q \in \mathcal{N}_i \tag{8}$$

$$x_{ij}^k \le x_{i'j'}^l, \forall i,j,i',j' \in \mathcal{M}, \mathcal{P}_{i'j'}^l \subseteq \mathcal{P}_{ij}^k \tag{9}$$

$$\sum_{i\in\mathcal{M}} \sum_{k\in\mathcal{K}_{ij}} x_{ij}^k \left(x_{ij}^k + \sum_{q\in\mathcal{N}_i} y_{qi} \right) \le N_{\text{th}}, \forall j \in \mathcal{R} \tag{10}$$

The intuition of these constraints can be interpreted as follows:

(4) Every MD has a valid primary path to some candidate roots.

(5) Each MD_i has at most one path to another MD_j.

(6) Every MD has exactly one backup parent. Notice that constraints (4) and (5) already guarantee every MD has exactly one primary parent.

(7) The primary parent and backup parent cannot be identical.

(8) The primary parent and backup parent cannot be on the same tree.

(9) The selected paths should together form a forest of trees. This constraint ensures that if a path $\mathcal{P}_{ij}^k$ is selected, its subpath $\mathcal{P}_{i'j'}^l$ must be selected. Therefore, the computed graph will be a tree.

(10) As described before, the number of MDs in the same tree is upperbounded by a threshold N_{th}. This security constraint considers both primary children and backup children in a tree.

C. Algorithm

The RDCTP is an NP-hard mixed-integer programming problem. To solve RDCTP, we propose a Two-step Heuristic:

1. Compute primary trees subject to the security constraint (10).
2. Compute backup links based on primary trees obtained from step 1.

Following the Two-step Heuristic, we propose a greedy algorithm based on Prim's Algorithm [14]. The algorithm is described in Algorithm 1.

The algorithm takes the MD topology, candidate roots and security threshold as input, and outputs a primary data collection forest $\mathcal{F}$ and a map of backup links $\mathcal{L}$, where $\mathcal{L}[x] = y$ if y is selected as the backup parent of x.

Algorithm 1: Two-step Greedy Algorithm

input : MD topology $\mathcal{G}$, a list of candidate roots $\mathcal{R}$, a security threshold N_{th}
output: Primary forest $\mathcal{F}$, backup links $\mathcal{L}$

```
   // Initialization
1  F ← EmptyTree
2  L ← EmptyMap
   // Compute primary forest by extended
   Prim's Algorithm
3  while exist some non-root node not in F do
4  |  x ← ExtendedPrimIter()
5  |  Add x to F
6  |  r ← TreeRootOf(x)
7  |  if r.size() ≥ N_th then
8  |  |  R.remove(r)
9  |  end
10 end
   // Select backup parent for each MD
11 foreach non-root node x do
12 |  y ← SelectBackup(x)
13 |  L[x] ← y
14 |  r ← TreeRootOf(y)
15 |  if r.size() ≥ N_th then
16 |  |  R.remove(r)
17 |  end
18 end
19 return F, L
```

For the first step, it computes the primary forest using an extended Prim's Algorithm. The Prim's Algorithm is generally used to compute minimum spanning tree given a network topology. In each iteration, it selects a node with minimum weight and adds it into the spanning tree.

Inspired by the Prim's algorithm, we group all the candidate roots as a root set $\mathcal{R}$, and define ExtendedPrimIter() (line 4) to find a non-root node x such that it introduces minimum extra latency to the tree roots. One node is added to the forest per iteration until all the non-roots are in the forest $\mathcal{F}$. After each iteration, if any tree root r has N_{th} nodes in its tree, we remove r from $\mathcal{R}$ so that no more non-root node can be assigned to tree r.

A similar approach is applied to construct backup links. For each non-root node, SelectBackup(x) (line 12) enumerates all the neighbors of x, and selects backup parent y on a different tree of x such that attaching x to y introduces minimum extra latency to the tree roots. After adding a backup link, if the size of the tree of the backup parent has reached the threshold, remove this root from root set.

VI. Simulation

We use the dataset of Utility Poles from Washington DC [15], which describes the geo-location information for more

than 48,000 utility poles. We extract multiple portions from the whole topology of size n, and randomly select m MDs as candidate roots. The communication delay between two MDs is defined proportional to their distance. In addition, we choose the communication range of each MD to be identical as $100m$. Since the overall density of poles is $271.18/km^2$, each MD has 8.5 neighbors on average.

We randomly generate samples of sub-portions from the dataset. The sizes of these samples vary from 500 to 1,400 nodes. For each sample size, we generate 10 different topologies and run simulations on them.

As described in the formulation, the size of each tree is upper-bounded by the security parameter N_{th}. In the case where N_{th} is relatively small to the total number of MDs n, we select m candidate roots such that $m = \frac{n}{N_{\text{th}}}\log(n)$.

Therefore, with high probability, the data collection forest can be constructed successfully.

We use the total data collection time with one failure as our evaluation metric. After the data collection forest is constructed, we select an arbitrary node to fail and turn on backup links for its children, and measure the total data collection time. The data collection time demonstrated in our result is normalized.

First, we compare the performance of Two-step Greedy Algorithm with a base line referred as *Random Algorithm*. In the random algorithm, each MD randomly selects two distinct neighbors as its primary and backup parents. We guarantee that the primary paths generated by the random algorithm also follows the constraints mentioned in Section V. We assume the leakage probability of each MD to be $p_{\text{leak}} = 0.01$ and set the security parameter as $N_{\text{th}} = 20$. Therefore, the probability of leaking a group key from each tree is $P_{\text{leak}}(\mathcal{T}) = 1-(1-0.01)^{20} = 0.182$.

The result is shown in Figure 5. The data collection time of Two-step Greedy Algorithm is significantly shorter than that of random approach by around 70% on average.

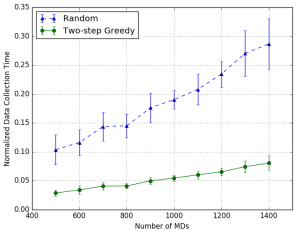

Fig. 5. Comparison to Base Line.

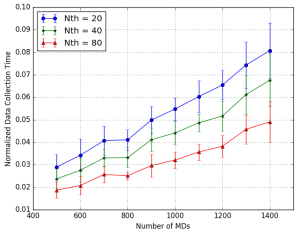

Fig. 6. Performance-Security Tradeoff.

Then, we run another simulation on the same topology samples. We compare the data collection time with one failure of Two-step Greedy Algorithm under different security parameter N_{th}. We set N_{th} to be 20, 40 and 80. Thus, the leakage probability of a data collection is 0.18, 0.33 and 0.55 respectively. It can be observed from Figure 6 that there is a tradeoff between data collection time and security parameter. Data can be collected faster if weaker security assumption is required.

VII. Conclusion

In this paper, we propose a resilient data collection mechanism in Smart Grid network. We propose a resilient data collection protocol to tolerate one-node failure in SG network. We formulate the Resilient Data Collection Tree Problem to optimize data collection time under security constraint, and design a Two-step heuristic to solve the problem. The simulation results show that our algorithm reduces the data collection time significantly.

References

[1] Q. Chen, D. Kaleshi, and Z. Fan, "Reconsidering the smart metering data collection frequency for distribution state estimation," in *Proc. of IEEE SmartGridComm*, 2014.

[2] H. Gao, X. Fang, J. Li, and Y. Li, "Data collection in multi-application sharing wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 2, Frebuary 2015.

[3] J. Silber, S. Sahu, J. Singh, and Z. Liu, "Augmenting overlay trees for failure resiliency," in *Proc. of IEEE Globecom*, 2004.

[4] M. D. Francesco, S. K. Das, and G. Anastasi, "Data collection in wireless sensor networks with mobile elements: A survey," *ACM Transaction of Sensor Networks*, vol. 8, no. 1, August 2011.

[5] R. Zhang, J. Pan, D. Xie, and F. Wang, "NDCMC: A hybrid data collection approach for large-scale wsns using mobile element and hierarchical clustering," *IEEE Internet of Things Journal*, to appear.

[6] Z. Xu, L. Chen, C. Chen, and X. Guan, "Joint clustering and routing design for reliable and efficient data collection in large-scale wireless sensor networks," *IEEE Internet of Things Journal*, to appear.

[7] A. Mehrabi and K. Kim, "Maximizing data collection throughput on a path in energy harvesting sensor networks using a mobile sink," *IEEE Transactions on Mobile Computing*, vol. 15, no. 3, March 2016.

[8] C. Wang, S. Guo, and Y. Yang, "An optimization framework for mobile data collection in energy-harvesting wireless sensor networks," *IEEE Transactions on Mobile Computing*, toappear.

[9] Y. Yao, Q. Cao, and A. V. Vasilakos, "Edal: An energy-efficient, delay-aware, and lifetime-balancing data collection protocol for heterogeneous wireless sensor networks," *Networking, IEEE/ACM Transactions on*, vol. 23, no. 3, pp. 810–823, 2015.

[10] X.-Y. Liu, Y. Zhu, L. Kong, C. Liu, Y. Gu, A. V. Vasilakos, and M.-Y. Wu, "Cdc: Compressive data collection for wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 8, August 2015.

[11] K. Wang, X. Qiu, S. Guo, and F. Qi, "Fault tolerance oriented sensors relay monitoring mechanism for overhead transmission line in smart grid," *Sensors Journal, IEEE*, vol. 15, no. 3, pp. 1982–1991, 2015.

[12] J. Kamato, L. Qian, W. Li, and Z. Han, "Biconnected tree for robust data collection in advanced metering infrastructure," in *Proc. of IEEE WCNC*, 2015.

[13] H. Jin, S. Uludag, K.-S. Lui, and K. Nahrstedt, "Secure data collection in constrained tree-based smart grid environments," in *Proc. of IEEE SmartGridComm*, 2014.

[14] R. C. Prim, "Shortest connection networks and some generalizations," *Bell system technical journal*, vol. 36, no. 6, pp. 1389–1401, 1957.

[15] DCGISopendata. (2002) Dcgis open data: Utility and communication. [Online]. Available: http://opendata.dc.gov/datasets/52a70a0438dc44818e97593d13d808ae_13