Available online at www.sciencedirect.com
ScienceDirect
Ad Hoc Networks 5 (2007) 100–111
Ad Hoc Networks
www.elsevier.com/locate/adhoc

Secure data aggregation without persistent cryptographic operations in wireless sensor networks ☆

Kui Wu [a,*], Dennis Dreef [a], Bo Sun [b], Yang Xiao [c]

[a] Department of Computer Science, University of Victoria, Victoria BC, Canada V8W 3P6
[b] Department of Computer Science, Lamar University, Beaumont, TX 77710, United States
[c] Department of Computer Science, University of Alabama, 101 Houser Hall, Box 870290, Tuscaloosa, AL 35487-0290, United States

Available online 23 June 2006

Abstract

In-network data aggregation is an essential operation to reduce energy consumption in large-scale wireless sensor networks. With data aggregation, however, raw data items are invisible to the base station and thus the authenticity of the aggregated data is hard to guarantee. A compromised sensor node may forge an aggregation value and mislead the base station into trusting a false reading. Due to the stringent constraints of energy supply and computing capability on sensor nodes, it is challenging to detect a compromised sensor node and keep it from cheating, since expensive cryptographic operations are unsuitable for tiny sensor devices. This paper proposes a secure aggregation tree (SAT) to detect and prevent cheating. Our method is essentially different from other existing solutions in that it does not require any cryptographic operations when all sensor nodes work honestly. The detection of cheating is based on the topological constraints in the aggregation tree. We also propose a weighted voting scheme to determine a misbehaving node and a secure local recovery scheme to avoid using the misbehaving node.

Keywords: Cheating detection; Wireless sensor networks; Data aggregation

1. Introduction

Extremely small sensors have been used broadly for various applications such as habitat monitoring, battlefield surveillance, and forest fire monitoring. A large number of tiny sensors collect measurement data and rely on multihop short range radio communication to send data to the processing center, which is also called the base station or the sink node. The major motivation to use tiny sensors is to save energy consumption and reduce system cost, as recent tiny sensors are expected to work without recharging for several months. The lifetime of sensors, however, depends on effective energy saving strategies such as sensor scheduling and in-network information processing to reduce the data traffic to the base station. One important type of in-network processing is data aggregation.

☆ This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), Canada Foundation for Innovation (CFI), and the Texas Advanced Research Program under grant 003581-0006-2006.
* Corresponding author.
E-mail addresses: wkui@uvic.ca (K. Wu), ddreef@uvic.ca (D. Dreef), bsun@cs.lamar.edu (B. Sun), yangxiao@ieee.org (Y. Xiao).

1570-8705/$ - see front matter
doi:10.1016/j.adhoc.2006.05.009

Depending on different application contexts, data aggregation could be in the form of data compression or the calculation of some statistical values, such as the mean, max, or min. While data aggregation can effectively reduce the amount of data transmitted to the base station, raw data items may be invisible to the base station and thus their authenticity and integrity are hard to guarantee. As such, data aggregation is potentially vulnerable to attackers who may inject bogus information or forge aggregation values without being detected. It may have a disastrous impact if end users respond according to the faulty information.

Some methods have been proposed to solve the above problem [2,3,7,10,12]. Existing methods depend on complex data authentication operations [2,3,12] or the statistical features of specific aggregation operations [1,9,10]. To guarantee correctness, *persistent* authentication operations are used in most existing methods. *Persistent* authentication, though very safe, may consume too much energy and is thus undesirable for sensor networks.

In this paper, we assume the existence of a secure mechanism but use it only when necessary, i.e., only when cheating activities are detected. This strategy is fundamentally different from existing solutions in that its security is based on cheating detection instead of persistent data authentication. This strategy can reduce energy consumption and more importantly can save the CPU resource. It, however, requires the support of a lightweight and scalable cheating detection method, which is a quite challenging task.

This paper is motivated to solve the aforementioned problem and has made the following contributions. First, it introduces topological constraints when building a secure aggregation tree (SAT), to facilitate the behavior monitoring of each aggregation sensor node. Second, when the aggregation values from an aggregation node are in doubt, a weighted voting scheme is proposed to decide finally whether the aggregation node is properly behaving or is cheating. Third, if a misbehaving node is detected, a local recovery scheme is presented to re-build SAT so that the misbehaving node is excluded from the aggregation tree. Finally, analysis and simulation are performed to demonstrate the effectiveness of SAT. Since no cryptographic operations are required when all nodes work honestly, our method is lightweight. Since no centralized operations are needed at the base station, our method also scales very well.

2. Assumptions

Two sensor nodes are called one-hop neighbors if they can communicate directly with each other and two-hop neighbors if they can communicate via two-hop radio transmission. In this paper, neighbors by default are referred to one-hop neighbors unless we explicitly indicate two-hop neighbors.

We assume that *a node cannot impersonate its neighbors*. This is not a strong assumption since if a node, A, is in the promiscuous listening mode, it can quickly detect any transmission from its neighboring nodes with A's identity. This assumption is to exclude the possibility that a node impersonates its father node in the aggregation tree to send false aggregation data such that an honest father is voted out of the network.

We assume that two neighboring sensor nodes have mechanisms for secure communication so that they can decrypt and authenticate each other's messages. There are a lot of key distribution methods proposed for sensor networks that can meet this requirement [4,6,11]. This seemly strong assumption does not mean that our method will always use message encryption and authentication. In contrast, they are used *only when cheating behavior is detected*. If all sensor nodes work honestly, no cryptographic operations are required.

3. Building a secure aggregation tree (SAT)

3.1. The structure of SAT

If we could build an aggregation tree such that any child node can monitor the behavior of its father node, the cheating activities of any non-leaf (aggregation) nodes can be detected. In this paper we do not consider the cheating behavior of leaf nodes since it is indistinguishable between cheating and malfunctioning for a leaf node. Obviously, to allow a child node to monitor its father node's behavior, it is required that the child node should be able to know all the messages from its sibling nodes to the father node. In other words, a father node together with its child nodes should form a *clique*.

3.2. A distributed algorithm to build SAT

In this section, we propose a distributed algorithm to build SAT with the assumption that each node knows its one-hop and two-hop neighbors.

The one-hop neighbors can be easily found with beacon messages, and the information of two-hop neighbors can be found with one local broadcast from each sensor node, indicating who are its one-hop neighbors.

The distributed algorithm builds the aggregation tree starting from the sink node and includes four steps as follows:

Step 1: The sink node locally broadcasts an *invitation* message to all of its one-hop neighbors, indicating that they should be its children. The *invitation* message includes the IDs of all nodes that a father node wants to invite to join the aggregation tree as its children. It should also include the hop count value to make a node aware of its minimal hop count to the sink node. The hop count value in the *invitation* message from the sink node is set to zero. A node sets its initial hop count value to infinity (i.e., a large integer like 10,000) and updates its hop count value as one plus the minimal hop count value in all received *invitation* messages.

Step 2: Once a node receives an *invitation* message, if this node has not joined the aggregation tree and the *invitation* message includes this node as a child node, then this node joins the aggregation tree and records the sender of the *invitation* message as its father node. It locally broadcasts a *join* message to notify its neighbors about this decision. This *invitation* message is also called *activating invitation* message since it requires the node to join the aggregation tree. Once a node joins the tree, later received *invitation* messages will be recorded for future use if the hop count value in the *invitation* messages is smaller than the node's current hop count value. More specifically, these *invitation* messages will be used to select a candidate father node if a current working father node is compromised. To avoid forming a poor aggregation tree, an additional rule should be applied, although this rule is rarely used if the network density is high: if a node receives an invitation message but the hop count value included in the message is 2 hops larger than its current hop count value, then this invitation message is ignored. We call this rule the *optimization rule* since it excludes the possibility of creating a poor aggregation tree that requires large energy consumption on data delivery.

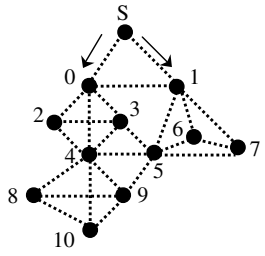
Step 3: After a node joins the aggregation tree, by checking its one-hop and two-hop neighbors, excluding those sibling nodes indicated in the *activating invitation* message, it can identify all the cliques that it belongs to. If such cliques cannot be found, then this node works as a leaf node. Otherwise, it selects the maximal clique and locally broadcasts an *invitation* message with the hop count value increased by one, indicating that all other nodes in the selected clique should be its children.

Step 4: Repeat step 2 and step 3 until all non-isolated nodes have joined the tree.

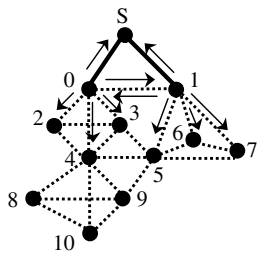
Note that if a node is disconnected from the sink node, it will not receive any *invitation* message and will not join the tree. In this case, the node is an isolated node and cannot be used by any means. As an example, Fig. 1 illustrates how an aggregation tree is built step by step.

Due to the topological constraint that an aggregation node together with its children should form a clique, it is possible that some nodes may not join the aggregation tree even if they have paths to the sink node. We call such nodes *sparse nodes* since they have only sparse set of neighboring nodes. Nevertheless, as demonstrated later, the ratio of the number of sparse nodes over the total number of sensor nodes is *extremely small* if the network density is reasonably high. As such, we could simply require the sparse nodes to send their messages to the sink node without performing any in-network processing. Any secure routing and secure unicast mechanisms could be applied to the messages sent from the sparse nodes.

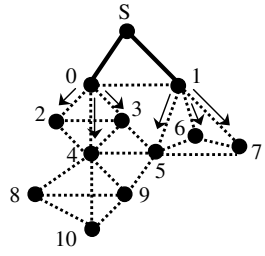
It is possible that in Step 2 and Step 3 local broadcast messages may collide and the correct information may not be received by receivers. Fortunately, this problem can be easily avoided in our tree buildup process, since the order of the broadcast messages from the children nodes can be scheduled by the father node. For instance, when a father node makes the selection of its children, it can arrange an arbitrary order for the children nodes' broadcasts and piggyback this information in the *invitation* message. Each child node is permitted to broadcast only in its allocated timeslot. Furthermore, to reduce broadcast overhead, a node may combine the *join* message and the *invitation* message into one single broadcast.



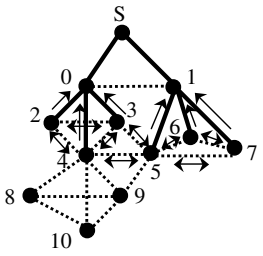
(a) S broadcasts an *invitation* message to nodes 0 and 1.



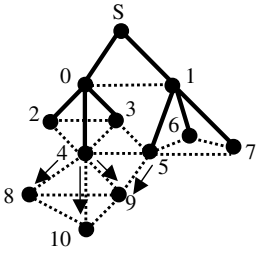
(b) Nodes 0 and 1 join the aggregation tree and broadcast a *join* message.



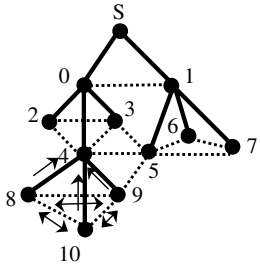
(c) Node 0 finds the maximal clique {0, 2, 3, 4} and invites nodes 2, 3, 4 to join the aggregation tree. Similarly, Node 1 finds the maximal clique {1, 5, 6, 7} and invites nodes 5, 6, 7 to join the aggregation tree.



(d) Nodes 2, 3, and 4 broadcast *join* messages to indicate node 0 as their father. Similarly, nodes 5, 6, and 7 broadcast *join* messages to indicate node 1 as their father.



(c) Node 4 finds the maximal clique {4, 8, 9, 10} and invites nodes 8, 9, 10 to join the aggregation tree. Similarly, Node 5 finds the maximal clique {5, 9} and invites node 9 to join the aggregation tree.



(e) Nodes 8, 9, and 10 broadcast *join* messages to indicate node 4 as their father. Node 9 records the *invitation* message from node 5 for future use if we assume that the *invitation* message from node 4 arrives first.

Fig. 1. An illustration of the steps in building up the SAT.

4. Cheating detection for data aggregation

4.1. Detection of potential misbehavior

The cheating detection is very similar to the *watchdog* introduced in [5], where each node works in the promiscuously listening mode to monitor all transmissions within its maximal radio range. With *watchdog*, each node, after sending a packet to its next hop node, listens to the channel to check if its next hop node relays the packet correctly. Similarly, in our method, due to the topological constraints in SAT, each node can overhear all messages sent to its father node and can monitor the message sent from its father node to its grandfather node to check if the father node performs data aggregation correctly. If a node's father node sends out a value that is *significantly* different from a correct aggregation value, the node will raise an *alert*. The criterion of setting a proper threshold value for raising an alert will be discussed in Section 5. In this paper, we differentiate an *alert* message from a *detection-confirmation* message. An *alert* message means a potential compromise, while a *detection-confirmation* message indicates the final confirmation of the existence of a cheating node.

Ideally, if a sensor node can overhear all messages sent to its father and track the values that have been aggregated, the above cheating detection will have low false positive ratio. In practice, however, it is possible that some messages to the father node are lost or the father node may not use exactly the same set of values for aggregation due to time asynchrony. In both cases, cheating detection with SAT may generate false alarms. The false alarm rate is obviously dependant on the specific application context and the criterion of raising alerts.

To draw a consistent conclusion about an aggregation node's behavior, we assign each *alert* with a

confidence value. This confidence value is calculated based on the specific aggregation function and is used to indicate the likelihood that the aggregation node is cheating. In the following, we propose a weighted voting method to make the final decision regarding whether or not an aggregation node is cheating.

4.2. Weighted voting

First of all, we want to remark that all control messages in the weighted voting process and the local recovery process should be encrypted and authenticated with some underlying secure communication mechanisms. This is to guarantee the correct final decision once the potential misbehavior is detected. Also, we stress again that *the secure communication is required only when an aggregation node is working improperly*.

Once a sensor node detects that its father node might be cheating, it sends out an *alert* message to all its neighbors except the father node. To keep a compromised node from sending out fake alert messages and also to keep the voting process invisible to the father node, the alert message should be authenticated. Note that the voting process should be secure, since otherwise an attacker can compromise a node and paralyze its father node as well by sending out fake alerts.

An *alert* message should include the cheating node's ID, the detecting node's ID, and the confidence value of the alert. The confidence value indicates the likelihood of a correct detection. A large confidence value indicates that the aggregation node is more likely to be cheating.

Once receiving *alert* messages from neighboring nodes, a node first checks if the cheating node is its father node. If yes, it calculates the weighted confidence value using the following formula:

$$F = \frac{\sum_{i=1}^{m_1} f_i}{m}$$

where f_i is the confidence value included in the alert message from the sibling node i, m is the total number of sibling nodes, and m_1 is the number of sibling nodes that send out an *alert* message.

If the weighted confidence value F is larger than a predefined threshold value, the father node is assumed to be cheating, and a *detection-confirmation* message will be broadcasted within a given hop limit. Any node receiving the *detection- confirmation* message will use the following local recovery mechanism to avoid using the compromised node.

4.3. Local recovery

If a sensor node, A, receives a *detection-confirmation* message, it checks if the cheating node is one of its child nodes or its father. If the cheating node is one of its child nodes, A will ignore any messages from that child.

If the cheating node is its father node, A will select another candidate father from its recorded *invitation* messages. Note that deadlocks cannot be formed since a node only records *invitation* messages that have smaller hop count values than its current hop count value. In other words, sensor node A selects another neighboring sensor node, B, as its father node *only if* B has a smaller hop count value to the sink node than A. This is to avoid the deadlock situation where two sibling nodes select each other as the potential father node. If a candidate father node cannot be found, then the current node becomes an isolated sensor node since none of its fathers can be used for reliable aggregation.

The newly formed "isolated" sensor nodes actually have a physical path to the sink node, although the path has to include some misbehaving nodes and may not be secure. If the sensor network still requires the sensing data from these "isolated" sensor nodes, more secure mechanisms such as data signature and authentication must be utilized for each message from the newly formed "isolated" nodes.

5. Examples of raising alerts

5.1. Problem modeling

Obviously, the selection of a proper threshold value to decide whether a father node is cheating relies on the application context and the aggregation function. *There is no generic criterion suitable for all scenarios.* In this section, we introduce the criterion on selecting the proper threshold value for several broadly used aggregation functions, namely, mean, max, and min.

There are mainly two reasons that the aggregation value at a father node is different from that calculated at a child node. The first is that some packets sent to the father node may be lost, and the other is that the father node uses a slightly different set of values for aggregation due to time asynchrony. In our model, a father node knows the number of its child nodes. When an aggregation calculation is required, the father node will use the most recently received values from its child nodes as the input.

That is, *we do not assume that data transmissions are well synchronized*. So a child node may not know exactly the data aggregated in the father node.

Assume that the child nodes, $C_1, C_2, \ldots, C_m$, send data packets to the father node, F. Assume that during a time period each child node $C_i (i = 1, \ldots, m)$ sends n packets to the father node, denoted as $D(C_i,1), D(C_i,2), \ldots, D(C_i,n)$ respectively. Due to packet losses or time asynchrony, the value of the aggregation function calculated by the father node might be any value in the set $\{f(D(C_1,j_1), D(C_2,j_2), D(C_3,j_3), \ldots, D(C_m,j_m))\}$ where f is the aggregation function and j_k $(k = 1,2,\ldots,m)$ might be any value in $\{1,2,\ldots,n\}$.

For a child node, when it monitors its father's behavior, it calculates the same aggregation function

$$f(\overline{D(C_1,j_1)}, \overline{D(C_2,j_2)}, \overline{D(C_3,j_3)}, \ldots, \overline{D(C_m,j_m)})$$

where $\overline{D(C_i,j_k)} (i = 1, \ldots, m)$ is the newest data received from the sibling node i. As a reasonable assumption, we assume that $|D(C_i,j_k) - \overline{D(C_i,j_k)}|$ is bounded by δ, since it is unlikely that during a short time period readings from the same sensor change dramatically.[1] We also assume that data from different child nodes are independent.[2]

Our problem could then be modeled as follows. Given an aggregation function f, an arbitrary positive value λ, a set of values $D_1, D_2, \ldots, D_m$, and another set of values $\overline{D_1}, \overline{D_2}, \ldots, \overline{D_m}$ with each $\overline{D_i}$ randomly selected from $[D_i - \delta, D_i + \delta]$ $(i = 1, 2, \ldots, m)$ respectively, what is the probability that $|f(D_1, D_2, \ldots, D_m) - f(\overline{D_1}, \overline{D_2}, \ldots, \overline{D_m})| \geqslant \lambda$? We denote the above probability as P_c.

The value of λ indicates the absolute difference between the aggregation value calculated at a father node and that monitored at a child node. Given a small threshold value θ, if $P_c \leqslant \theta$, then the child node should raise an alert since it is unlikely (i.e., P_c is too small) that the father node should generate such a different aggregation value. The confidence value for the alert is set to $1 - P_c$.

5.2. Mean

Denote the mean of $D_1, D_2, \ldots, D_m$ as $S = \frac{\sum_{i=1}^{m} D_i}{m}$. Also denote the mean of $\overline{D_1}, \overline{D_2}, \ldots, \overline{D_m}$ as $\overline{S} = \frac{\sum_{i=1}^{m} \overline{D_i}}{m}$, where $\overline{D_i}$ is randomly selected from $[D_i - \delta, D_i + \delta]$ respectively. Note that S is a constant and $\overline{S}$ is a random variable. We are asked to estimate $P_c = Pr(|\overline{S} - S| \geqslant \lambda)$.

It is obvious that when $\lambda > S + \delta$ or $\lambda < S - \delta$, $P_c = 0$.

When $S - \lambda \leqslant S \leqslant S + \lambda$, by Chebyshev's bounds we have

$$P_c \leqslant \frac{\mathrm{VAR}[\overline{S}]}{\lambda^2}.$$

Since $\overline{D_i}$, $i = 1, 2, \ldots, m$ are independent, it is easy to calculate that

$$\begin{aligned}
\mathrm{VAR}[\overline{S}] &= \frac{1}{m}\sum_{j=1}^{m} \mathrm{VAR}[\overline{D_i}] \\
&= \frac{1}{m}\sum_{j=1}^{m} (E[\overline{D_j}^2] - (E[\overline{D_j}])^2) \\
&= \frac{1}{m}\sum_{j=1}^{m} \left(\frac{1}{2\delta} \int_{D_j-\delta}^{D_j+\delta} \overline{D_j}^2 d(\overline{D_j}) - D_j^2 \right) \\
&= \frac{1}{3}\delta^2.
\end{aligned}$$

In conclusion,

$$P_c \begin{cases} = 0 & \text{if } \lambda > S + \delta \text{ or } \lambda < S - \delta \\ \leqslant \frac{\delta^2}{3\lambda^2} & \text{otherwise} \end{cases}.$$

5.3. Min/Max

Denote the min of $D_1, D_2, \ldots, D_m$ as $m = \min\{D_1, D_2, \ldots, D_m\}$. Also denote the min of $\overline{D_1}, \overline{D_2}, \ldots, \overline{D_m}$ as $\overline{m} = \min\{\overline{D_1}, \overline{D_2}, \ldots, \overline{D_m}\}$, where $\overline{D_i}$ is randomly selected from $[D_i - \delta, D_i + \delta]$ respectively. Note that m is a constant and $\overline{m}$ is a random variable. We are asked to estimate $P_c = Pr(|\overline{m} - m| \geqslant \lambda)$.

For the function of *min*, we can get an accurate formula instead of a bound to calculate the above probability. First, let us check under what situation the above probability is zero. It is easy to see that it is impossible for $\overline{m}$ to get a value smaller than $m - \delta$ or larger than $m + \delta$ if the aggregation node is honest. That is, if the aggregation node is honest, the inequality $-\delta \leqslant \lambda \leqslant \delta$ must be true. Therefore, we get if $|\lambda| > \delta, P_c = Pr(|\overline{m} - m| \geqslant \lambda) = 0$. In other words, if the absolute difference between the aggregation value calculated at the father node and that monitored at a child node is larger than δ, the child node is sure that the father node is cheating and raises an alert with 100% confidence.

[1] This phenomenon is also called temporal correlation of sensory data.

[2] With this assumption, we assume that the spatial correlation of data is negligible. Refer to Section 5.4 for further discussion.

Next, we calculate the probability $Pr(|\overline{m} - m| \geqslant \lambda)$ when $|\lambda| \leqslant \delta$.

For each random variable $\overline{D_i}, i = 1, \ldots, m$, the *cdf* (cumulative distribution function) of $\overline{D_i}$ is

$$F_i(x) = Pr(D_i < x) = \frac{x - D_i + \delta}{2\delta}.$$

Therefore, the *cdf* of $\overline{m}$ is

$$F(x) = 1 - \prod_{i=1}^{m}(1 - F_i(x)) = 1 - \prod_{i=1}^{m}\left(1 - \frac{x - D_i + \delta}{2\delta}\right).$$

By calculation,

$$\begin{aligned} P_c &= Pr(|\overline{m} - m| \geqslant \lambda) = Pr(\overline{m} - m \geqslant \lambda) \\ &\quad + Pr(\overline{m} - m \leqslant -\lambda) = 1 - F(m + \lambda) + F(m - \lambda) \\ &= 1 + \prod_{i=1}^{m} \frac{\delta + D_i - m - \lambda}{2\delta} - \prod_{i=1}^{m} \frac{\delta + D_i - m + \lambda}{2\delta}. \end{aligned}$$

In conclusion,

$$P_c = \begin{cases} 0 & \text{if} |\lambda| > \delta \\ 1 + \prod_{i=1}^{m} \frac{\delta + D_i - m - \lambda}{2\delta} - \prod_{i=1}^{m} \frac{\delta + D_i - m + \lambda}{2\delta} & \text{otherwise} \end{cases}.$$

The calculation of P_c for the aggregation function *max* is very similar to that for the *min*. To save space, we omit its calculation.

5.4. Further discussion

The analysis in previous sections only provides examples on cheating detection. For demonstration reason, we have made certain assumptions that (1) an aggregation node does not have data itself and (2) data from different children are independent or their correlation is negligible. While these assumptions hold true for some applications, it is not surprising that readers can easily find counter-examples. It is, however, unrealistic to propose a generic solution suitable in all situations.

If an aggregation node itself also has data to add in the aggregation function, the only way to detect a cheating aggregation node is to require the aggregation node to piggyback its own data together with the aggregation value. Since we use a clique-tree for aggregation, this add-on value is known by all children such that cheating detection can be performed.

If the spatial correlation in sensory data is not negligible, the above analytical results do not hold anymore and must be adjusted. Since the spatial correlation is usually positive[3], the actual bound will be looser. For instance, the variation of the mean function will be larger than that calculated above due to positive correlation. Nevertheless, it is impossible to quantify the correlation without considering dataset from a specific application. To summarize, accurate cheating detection criteria must consider specific application context, which is obviously beyond the focus of this paper.

[3] Data from different children *usually* have the same trend (increasing or decreasing) in real applications.

6. Cost analysis

6.1. Rationale

The rationale to use SAT is that energy could be saved by requiring no cryptographic operations when sensor nodes behave correctly. This rationale is justifiable only if (a) the energy cost on the monitoring with SAT in a long run is smaller than the energy cost on cryptographic operations and (b) SAT does not incur much larger energy cost on data transmission than other types of aggregation tree. In the following, we demonstrate that the two conditions hold true for SAT.

6.2. Energy cost on monitoring

In order to check if the energy cost of monitoring is smaller than traditional cryptographic operations, we compare the energy cost on computing aggregation functions, e.g., max, min, and mean, and that with a widely used cryptographic algorithm in sensor networks, RC5.

The amount of computational energy consumed by a function on a given microprocessor is primarily determined by the power consumption of the processor, the clock frequency of the processor, and the number of clocks needed by the processor to compute the function. We assume that energy consumption cannot be significantly reduced via a reduction in clock frequency. Since the function and the efficiency of its software implementation determine the number of clocks necessary to perform the function, the processing overhead in terms of the *Number of Basic Operations* can reflect the implementation efficiency and the energy consumption on computing a function.

Thus, we calculate the *Number of Basic Operations* of the aggregation functions and compare them with those of RC5. We consider Addition, XOR, Shift (1 bit), Fetch (fetch a value from the main memory to a register), and Store (store a value in a register to the main memory) as our basic opera-

tions. In particular, we choose RC5-32/12/X, (i.e., 32 bits words, 12 rounds, and X as the key length) based on the algorithm in [8]. The key-expansion routine is the most time-consuming part in running the RC5 algorithm. Because most wireless sensor networks that adopt RC5 as their underlying cipher require the S-Table to be computed in advance to speed up their sensors' operation, we do not consider the cost of computing the S-Table in our analysis.

The cost of performing one general n-bits multiplication is roughly as $\frac{n}{2}$ additions and $\frac{n}{2}$ shifts in average on n-bit registers. Since a division can be reduced to a multiplication, we use the same estimation for the division. Also, the same estimation is made to the general modulo. A multiplication by 2 is a left-shift operation; the operation of (n mod 32) is considered one XOR operation (in fact, we need a bitwise AND). In Rivest's algorithm, "shift B bits ($\lll B$)" means "shift (B mod 32) bits." Thus, there is a bitwise AND involved. Also, we use 16 as the average value of (B mod 32). For RC5, we assume that the values of A and B, the two words to be encrypted, remain in the registers during the course of computation.

To make the comparison plausible, we assume that the input data size to the aggregation function is the same as the input data size to RC5. The actual cost of the aggregation function should be smaller than that we calculate here, since RC5 is a block cipher that enforces the input data padded into blocks when the input data size is not divisible by the size of the block. Note that when we calculate the min/max among n data items, we require $n - 1$ times of comparison operation and the energy cost on each comparison operation is considered as the same as that on one Addition.

Table 1 depicts the comparison of the number of basic operations. It is obvious that aggregation operations are much simpler than cryptographic operations. This fact clearly justifies the energy efficiency of using SAT compared to using persistent cryptographic operations.

Table 1
Numbers of basic operations in RC5-32/12/X and aggregation functions

	RC5-32/12/X	Min/Max	Mean
8 bit Processor 8 byte Packet	4192	23	23
8 bit Processor 16 byte Packet	8384	47	95
8 bit Processor 32 byte Packet	16,768	95	105

6.3. Energy cost on data transmission

6.3.1. A note on radio transmission

The structure of SAT fully takes advantage of the broadcast feature of radio channels. Since a node together with its sibling nodes and its father node forms a clique, no extra energy is required for receiving packets from sibling nodes and the father node if the node is set to promiscuous listening mode. This is the same as the *watchdog* in [5].

6.3.2. Aggregation returning values with a fixed data size

We assume that the energy cost on data transmission is mainly decided by the amount of data transmitted. Although it is true that the transmission distance can also impact the energy cost, we ignore this impact since the radio range of sensor nodes is generally very small. Within a short range, the difference in the energy cost with different transmission distances may be negligible.

For most data aggregation functions such as min, max, and mean, the energy cost with different aggregation trees is roughly the same. This is because these aggregation functions return a constant number of bits no matter what the number of the input data items is. In this case, each sensor node, no matter it is a leaf node or an aggregation node, sends out the same amount of bits and thus has roughly the same energy consumption on data transmission. The energy cost on data transmission with any aggregation tree is $(n - 1) * e$ where n is the number of total sensor nodes and e is the energy cost on data transmission at each sensor nodes.

6.3.3. Aggregation returning values with variable data sizes

For some aggregation functions, the return value may not have a constant number of bits. Instead, the size of the return value is a function of the total size of input data. For example, data compression is this type of aggregation function. Under this situation, we are concerned whether or not SAT may incur extra data transmission overhead compared with other types of aggregation trees.

Note that data compression may not be an effective in-network data processing method since all information has to be processed in the sink node finally. Due to this reason, our analysis should be considered only as a guideline of cost estimation for *potential data aggregation functions* that may exist in the future. This is also the reason why we

do not perform analysis for compression-like aggregation in the previous section.

We assume an aggregation function that compresses the input data with a compression ratio of $\gamma, 0 < \gamma \leqslant 1$. We assume that all sensor nodes generate the same amount of measurement data, and each sensor node will aggregate the data reported from its children (if any) and the data generated by itself.

Without losing generality, we base our calculation on the assumption that each sensor generates 1 byte of measurement data. For a given aggregation tree, denote the maximal layer of the aggregation tree as L (the root node is layer-0) and the total number of nodes as n. Denote the number of layer-i nodes as l_i. Then the total transmission cost of an aggregation tree is:

$$C = \sum_{i=1}^{L} \sum_{j=1}^{i} \gamma^j l_i \geqslant \gamma(n-1) \tag{1}$$

where $l_1 + l_2 + \ldots + l_L + 1 = n$.

Theorem 1. *If the aggregation function compresses the input data with a constant compression ratio $\gamma(0 < \gamma \leqslant 1)$, the transmission cost of SAT is bounded by a constant approximate ratio to that of the optimal aggregation tree.*

Proof. Given a sensor node, A, assume that all paths between A and the sink node, S, are listed as $P_0, P_1, \ldots, P_m$ in the increasing order of the path length. Assume that the length of P_0 is l. Our proof is based on the fact that the optimal aggregation tree with the minimal communication cost must include A as its layer-l node (The root of the tree is layer-0, and the children of the root are layer-1, and so on.). This is because the cost for transmitting one byte from A is $\sum_{j=1}^{i} \gamma^j$ where i is the number of layer to which A belongs and γ $(0 < \gamma \leqslant 1)$ is the compression ratio. This is a monotonic function with i, and thus the higher the layer to which A belongs, the larger the communication cost of the aggregation tree.

If we assume that each local broadcast requires the same time, then the first received invitation message (not necessarily an activating invitation message) should come along the shortest path from the sink node. Assume that the father node of A along the path P_0 is B (There may exist multiple paths with the same path length as P_0, but we consider only one path for simplicity.). According to the tree buildup process of SAT, the following possibilities exist:

Case a: If A belongs to the maximal clique checked at node B, then A will receive an *activating invitation* message and join SAT. In this case, the cost of delivering messages from A with SAT will be the same as that with the optimal aggregation tree.

Case b: If A does not belong to the maximal clique checked at node B, then A is either a sparse node, which is a rare case from later simulation when network density is reasonably high, or A becomes a grandchild of B if one of B's children invites A as its child. A cannot become B's grand-grandchild since only two-hop neighbors are considered when building SAT and also because of the *optimization rule*, i.e., if a node receives an activating invitation but the hop count value included in the message is 2 hops larger than its current hop count value, then this invitation message is ignored. In this case, the cost of delivering one byte from A to the sink node is $\gamma^{l+1} (0 < \gamma \leqslant 1)$ more bytes compared to the optimal aggregation tree.

Case c: A does not become a descendant of B, but instead it becomes a child along other path $P_i, i > 0$. According to the *optimization rule*, the cost of delivering one byte from A to the sink node requires at most $\gamma^{l+2} + \gamma^{l+1} (0 < \gamma \leqslant 1)$ more bytes compared to the optimal aggregation tree. □

Therefore, according to Formula (1), in the worst case the ratio of energy cost of SAT (excluding the sparse nodes) over the energy cost of the optimal aggregation tree is

$$\begin{aligned} &\frac{\sum_{i=1}^{L}\sum_{j=1}^{i}(\gamma^{j+2} + \gamma^{j+1} + \gamma^j) l_i}{C} \\ &= \frac{C + \sum_{i=1}^{L}\sum_{j=1}^{i}(\gamma^{j+2} + \gamma^{j+1}) l_i}{C} \\ &\leqslant 1 + \frac{\gamma^2(\gamma+1)\sum_{i=1}^{L} l_i}{C} \leqslant 1 + \gamma^2 + \gamma. \end{aligned} \tag{2}$$

7. Simulation study

7.1. Simulation model

In this section, we perform simulation study to further demonstrate the feasibility and the effectiveness of SAT. The simulator was implemented in JAVA. We want to illustrate that SAT does not introduce extra transmission cost compared to other

commonly used tree structures and the proportion of the sparse nodes in SAT is extremely small for dense networks. In addition, we want to justify that the cost for local recovery is practically acceptable.

For comparison purposes, we implemented SAT and other types of tree structures, namely, BFT (Breadth-First Tree) and BFT-D (Breadth-First Tree with the Distance constraint). In BFT-D, we list node A as node B's child during the breadth-first search only when node A is further away from the sink than node B. The simulation was run on a variety of scenarios. The sensor nodes were deployed randomly in a 1000 m by 1000 m area with nodes having a transmission range of 50 m. We changed the number of sensor nodes from 500 to 5000 to change network density and placed the sink node in four different locations, i.e., the up-left corner $(0,0)$, the top-middle $(500,0)$, the left-middle $(0,500)$, and the center $(500,500)$. In order to get convincing results, we ran each scenario with three different random seeds. The simulation results were obtained by calculating the average of all runs.

Three performance metrics were considered:

1. The average depth of leaf nodes. It is calculated as the average hop count of all leaf nodes to the sink node. This metric is used to evaluate the average height of the aggregation tree and thus to estimate the cost for data transmissions with the tree.
2. The ratio of the sparse nodes. It is calculated as the ratio of the number of sparse nodes (refer to Section 3.2) over the total number of sensor nodes. Since sparse nodes cannot use SAT for their message transmission, a small number of sparse nodes is desirable.
3. The number of local candidate paths. It is calculated as the number of alternative paths from a node to its SAT grandparent node, i.e., paths without using the node's SAT father. Such paths could be used in local recovery (Section 4.3) and thus are called local candidate paths. We searched for the local candidate paths with limited number of hops.

7.2. Simulation results

7.2.1. The average depth of leaf nodes

Fig. 2 shows results of the average depth of the leaf nodes with different tree structures. From the results, the three tree structures, SAT, BFT, and BFT-D, have very close performance. The average depth of leaf nodes indicates the average height of the aggregation tree and thus the approximate cost on message delivery with the aggregation tree. Fig. 2 illustrates that SAT, despite its advantage for aggregation monitoring, does not incur obvious extra overhead on message delivery compared to other types of trees. This is because the *optimization rule* in SAT (refer to Section 3.2) excludes the possibility of creating a poor aggregation tree, and also because the inherent feature of radio broadcast enhances the chance of forming a clique structure locally. Another phenomenon is that the average depth of the leaf nodes increases when the number of nodes is increased from 500 to 1000, and then it remains roughly stable with the further increase of the number of nodes. This is because the network connectivity is poor when the number of nodes is smaller than 1000. When the network density is high enough, for instance, when the number of network nodes is larger than 2000, each node can find the shortest path to the sink node with the length roughly equivalent to the direct distance between the node to the sink node.

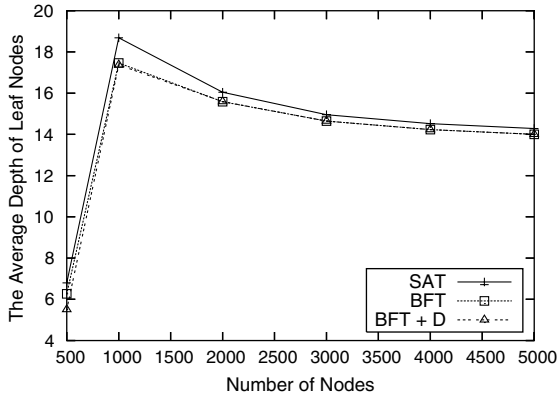


Fig. 2. Comparison of different tree structures.

7.2.2. The ratio of sparse nodes

Fig. 3 shows the ratio of the number of sparse nodes over the total number of nodes. From the figure, we can see that when the network is sparse, e.g., when the number of nodes is smaller than 1000, the number of sparse nodes is not negligible. Nevertheless, when network density becomes higher, the performance of SAT becomes significantly better with the ratio of sparse nodes quickly dropping to almost zero. This phenomenon justifies our previous claim that with dense networks, the cost on message delivery from the sparse nodes is negligible.

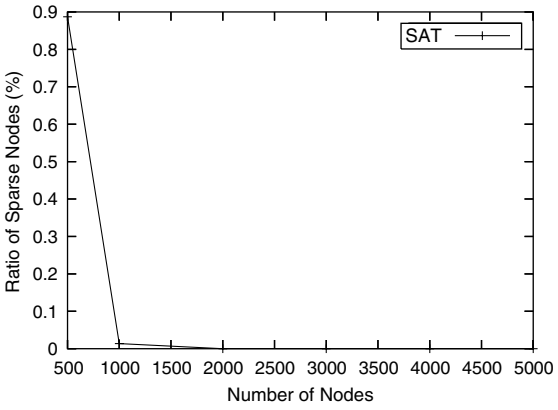


Fig. 3. The ratio of sparse nodes.

7.2.3. The number of local candidate paths

Table 2 shows the number of candidate paths from a node to its grandfather node without using its current father node on the SAT. Such candidate paths could be used for local recovery once an aggregation node is detected to behave abnormally. With a looser constraint on the hop count from a node to its grandparent, more candidate paths could be found and thus the chance of successful local recovery is higher. But a candidate path with larger hop count requires more message transmissions.

Table 2
Number of candidate paths

Nodes	1 Hop	2 Hops	3 Hops	4 Hops
500	0.0589	0.5722	3.2773	18.791
1000	0.0225	0.7615	7.9544	77.6229
2000	0.0033	0.8896	23.0084	444.9818
3000	0.0020	0.9331	45.5328	1323.001

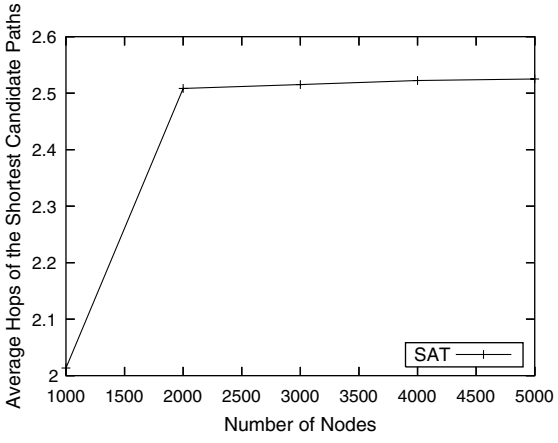


Fig. 4. Average hops of the shortest candidate paths.

There is a tradeoff between the possibility of local recovery and the recovery cost. We observe from the figure that with a hop count constraint of 3, more than 15 local candidate paths could be found in most cases. Also, from Fig. 4, the average hops of the shortest path from a node to its grandfather node without using its current father node is smaller than 3. This clearly demonstrates that the energy cost for local recovery is acceptable since local broadcast with hop limit of 3 suffices for the requirement in dense networks.

8. Conclusion

Traditional solutions to secure data aggregation use persistent data authentication that incurs heavy computational overheads even if no attackers or misbehaving nodes exist in the network. Such computational overheads waste energy and may greatly reduce the available CPU resource for data processing. At the worst case, some CPU-intensive applications, such as audio/video monitoring, may become impossible. An ideal security solution should not perform any cryptographic operations when the network works correctly, and uses data authentication only when misbehaving nodes are detected. For this purpose, we propose a secure aggregation tree (SAT) with which it is very easy to observe the behavior of aggregation nodes without resorting to persistent data authentication. Another benefit of the proposed SAT is that no additional routing algorithm is needed and the SAT automatically forms a routing approach. With analysis and simulation, we demonstrate the feasibility and effectiveness of using SAT to monitor the aggregation operations.

References

[1] A. Boulis, S. Ganeriwal, M.B. Srivastava, Aggregation in sensor networks: an energy – accuracy tradeoff, Elsevier Adhoc Networks J (special issue on sensor network protocols and applications), 2003.
[2] H. Cam, S. Ozdemir, P. Nair, D. Muthuavinashiappan, Espda: Energy-efficient and secure pattern-based data aggregation for wireless sensor networks, in IEEE Sensors 2003 Conference, Toronto, Canada, October 22–24, 2003.
[3] L. Hu, D. Evans, Secure aggregation for wireless networks, in Workshop on security and assurance in Ad hoc Networks, January 2003.
[4] C. Karlof, N. Sastry, D. Wagner, Tinysec: A link layer security architecture for wireless sensor networks, in Second ACM Conference on embedded networked sensor systems (SensSys 2004), November 2004.

[5] S. Marti, T.J. Giuli, K. Lai, M. Baker, Mitigating routing misbehavior in mobile ad hoc networks, in mobile computing and networking, 2000, pp. 255–65.
[6] A. Perrig, R. Szewczyk, V. Wen, D. Culler, J.D. Tygar, Spins: Security protocols for sensor networks, Wireless Networks 8 (5) (2002) 521–534.
[7] B. Przydatek, D. Song, A. Perrig, Sia: Secure information aggregation in sensor networks, in ACM SenSys 2003, November 2003.
[8] R. Rivest, The rc5 encryption algorithm, in Proceedings of 1st Workshop on Fast Software Encryption, 1995, pp. 86–96.
[9] N. Shrivastava, C. Buragohain, D. Agrawal, S. Suri, Medians and beyond: new aggregation techniques for sensor networks, in: SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems, ACM Press, New York, NY, USA, 2004, pp. 239–249.
[10] D. Wagner, Resilient aggregation in sensor networks, in ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN '04), October 2004.
[11] R. Watro, D. Kong, S. fen Cuti, C. Gardiner, C. Lynn, P. Kruus, Tinypk: securing sensor networks with public key technology, in: SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks, ACM Press, 2004, pp. 59–64.
[12] W. Zhang, G. Cao, Group rekeying for filtering false data in sensor networks: A predistribution and local collaboration based approach, in IEEE INFOCOM, March 2005.

Kui Wu received the Ph.D degree in computing science from the University of Alberta, Canada, in 2002. He then joined the Department of Computer Science, University of Victoria, Canada, where he is currently an Assistant Professor. His research interests include mobile and wireless networks, sensor networks, network performance evaluation, and network security.

Dennis Dreef is currently a Master's student in Computer Science at the University of Victoria. His research interests include Ad Hoc Networks, Wireless Sensor Networks, and Network Security.

Bo Sun received his Ph.D degree in Computer Science from Texas A & M University, College Station, USA., in 2004. He is now an assistant professor in the Department of Computer Science at Lamar University, USA. His research interests include the security issues (intrusion detection in particular) of Wireless Ad Hoc Networks, Wireless Sensor Networks, Cellular Mobile Networks, and other communications systems.

Yang Xiao worked at Micro Linear as an MAC (Medium Access Control) architect involving the IEEE 802.11 standard enhancement work before he joined Dept. of Computer Science at University of Memphis in 2002. He is currently with Department of Computer Science at The University of Alabama. He is an IEEE Senior member. His research interests include wireless security and wireless networks.