

Secure Feedback Service in Wireless Sensor Networks

Di Ma

University of California, Irvine
dma1@ics.edu.uci

Abstract. In many sensor network applications, it is critical for the base station to know the delivery (or execution) status of its broadcast messages (or commands). One straightforward way to do so is to let every sensor node send an authenticated acknowledgement (ACK) to the BS directly. However this naive solution is highly communication inefficient and may result in severe ACK implosion near the BS. In this paper, we propose a communication efficient scheme to provide secure feedback service in sensor networks. In our basic scheme, we use ACK aggregation to reduce the ACK traffic. Meanwhile we store audit information for each aggregation operation so that the BS can use the audit information to locate errors in the network. We further improve the basic scheme by constructing a balanced aggregation tree to reduce localization delay and using Bloom filters to reduce storage requirement in each sensor for storing audit information. We analyze the performance of the proposed scheme and show it achieves good bandwidth gain over the naive approach.

Keywords: Feedback Service, Sensor network, ACK aggregation, ACK implosion, MAC, Bloom Filter, Authentication.

1 Introduction

Wireless sensor networks (WSNs) enable data gathering from a vast geographical region and thus present unprecedented opportunities in a wide range of tracking and monitoring applications from both civilian and military domains. In a WSN, there exist hundreds or thousands of low-cost sensors which sense and collect data from the environment for some given tasks. The sensed data is then forwarded to the base station (BS) or sink in a hop-by-hop manner for further processing. A BS is a powerful control unit for WSNs which processes the data gathered by the sensors and operate the WSN by issuing commands/queries to the sensors. In many WSN applications, reliable data delivery is critical [18]. In these applications, the BS needs to know whether the sensors (the intended receivers) have received its broadcast/multicast messages or performed certain actions as it commanded.

For example, in a security application where image sensors are used to detect and identify the presence of critical targets [18], the BS may send one of the following three classes of messages, all of which have to be delivered reliably to the sensors and thus message delivery status is wanted: (i) Control-data. The BS may want to send a particular (say upgraded) image detection/processing software to the sensors which are configurable; (ii) Query-data. The BS may have to send a database of target images

to the sensors, to help them in the image recognition triggered by subsequent queries; (iii) Query. The BS may send out one or more queries requesting information about the detection of a particular target. The sensors can then match targets detected with the pre-stored images, and respond accordingly.

Another example of explicit feedback is the request for expected action acknowledgement. In some wireless sensor and actuator networks, the BS may send (broadcast/multicast) commands to the sensors periodically and expect the sensors to perform certain actions. For security purpose, the BS expects ACKs from sensors. In this way, the BS knows the current network status which helps the BS to monitor the whole network.

A third example of explicit feedback is to detect the Denial-of-Message (DoM) attack [14]. Message broadcast is a fundamental communication primitive in most sensor networks. Sensors in an adversarial environment might be deprived of broadcast messages under the DoM attack. To detect the existence of DoM attack, every broadcast recipient, the sensor, is required to send an authenticated ACK to the BS. If the BS did not receive an ACK from a sensor node in a certain period, it will assume that this sensor node is under DoM attack.

The above examples clearly clarify both the necessity and importance of a secure feedback service in some WSN applications. A naive solution is for each sensor node to send its authenticated ACK to the BS directly. However, this may result in ACK implosion near the BS as one broadcast message may result in thousands of ACKs. When thousands of ACKs are forwarded to the BS at the same time, ACK implosion will occur near the BS. At the same time, transmission of thousands of ACKs is very expensive. According to [2], wireless transmission of a bit can require over 1000 times more energy than a single 32-bit computation. Communication inefficiency associated with the naive solution also shortens the lifetime of the whole WSN. In the naive approach there is a widely differing data communication load amongst sensors at different levels. Sensors closer to the BS have to send significantly larger amounts of data than their descendants and hence they use up their batteries and die sooner. When a level of nodes closer to the BS stop functioning, then the whole WSN stops functioning as well. Therefore, nodes would have to either be swapped around manually or replaced upon failure, both tasks being quite impractical when considering the number of nodes at the various levels.

To prevent ACK implosion, we propose to use ACK aggregation to reduce the ACK traffic. In an ACK aggregation scheme, multiple ACKs are compressed into a single aggregated tag and verifying this aggregated tag equals to verify all the component ACKs. Related cryptographic primitives such as *multisignature* [15, 17, 4, 6] and *aggregate signature* [5, 12, 11] can be used to aggregate signatures. However these signature schemes are not suitable in the WSN setting because of their expensive public-key operations. In the symmetric key setting, Exclusive-Or (XOR) has been used in [14, 1] as the aggregation function to aggregate ACKs from sensors. However, the security of XOR-based aggregation is not clear (we discuss this in detail in Section 3.2). We propose to use collision resistant hash function to do the aggregation and the security of our hash-based aggregation scheme is based on the collision-resistance property of the underlying hash function.

Although ACK aggregation reduces ACK traffic, it loses detailed network information. When errors happen, the final aggregate arriving at the BS only tells the BS the fact that there is something wrong. It cannot tell the BS how many nodes are in problem if the BS wants to know the severity of the problem. Furthermore WSNs are usually location-aware in nature. Specifically, if data is obtained without the corresponding location information, the data may be useless. Sensors positioned in different geographical areas may have different application importance. In this case the BS may want to know the distribution of nodes in problem, whether it is network wide or just a small area of network failing in function. The aggregate simply cannot answer these questions for the BS.

In this paper, we propose a communication efficient scheme to provide secure feedback service in WSNs. In our scheme, sensors in adjacent area are implicitly grouped together through construction of an aggregation tree among all the group members. ACKs from group members are aggregated together and the final aggregate from the root of the aggregation tree is sent to the BS. To provide detailed status information to the BS when the final aggregate value fails in the verification process, audit information for aggregation operation are stored in intermediate nodes in the aggregation tree. The audit information allow the BS to isolate faults in the network. Therefore our scheme consists of two main functioning parts: ACK aggregation and audit, fault localization. These two functioning parts work complementarily to provide detailed feedback service to the BS.

Our aggregation scheme is different from data aggregation [10, 8, 20, 7, 21]. In our scheme, authentication objects of messages, MACs, are aggregated while individual messages are kept intact. In a data aggregation scheme, individual data information is lost and the aggregate gives statistical information like MAX, MIN, AVG or Median to the verifier. Data aggregation schemes cannot be used in applications, for example, temperature developing pattern sensing in a nuclear reactor, which require the presentence of individual data record.

The remaining part of this paper is organized as follows: Section 2 formulates the system assumptions and security model. Section 3 presents our basic scheme and Section 4 gives improvements. Section 5 analyzes the proposed scheme. We conclude the paper in Section 6.

2 System Assumptions and Security Model

We assume a general WSN with n sensors and a single BS. We assume that all n nodes are alive. As sensors are unusually put in an unattended and adversary environment because of application nature, we require end-to-end security a must: a node's ACK should be able to uniquely identified by the BS and no one can cheat the node's status. In our scheme, we assume each sensor node has a unique identifier S and it generates its ACK in the form of a MAC with a unique secret symmetric key K_S shared with the BS. Thus we assume a secure key-management protocol is present to establish and manage the secure pair-wise key between each node and the BS.

We further assume the existence of a broadcast authentication primitive such that every node will receive the broadcast message from the BS in an authenticated fashion.

This broadcast authentication could, for example, be performed using μ TESLA [19] or the one-time-signature based broadcast authentication scheme proposed in [9]. To prevent re-play attack, there is a nonce or a unique message ID associated with each broadcast message requesting acknowledgement. A sensor node replies the BS with its MAC after it receives the broadcast message. All ACKs are supposed to arrive at the BS within a fixed period.

Nothing can be made to tell whether an authenticated ACK is generated by a node or an attacker who obtains the secret key of the node. Instead we assume only a small portion of the total number of sensors can be compromised and their secret keys are exposed to the attacker. The attacker has a network-wide presence and can record, modify, inject or delete at will. The goal of an attacker is to report falsified information in order to hide the real network status information from the BS. The attacker may forge a bogus ACK of a sensor node which cannot respond because of being denied from receiving broadcast messages; the attacker may modify the ACK aggregate and pass the altered aggregate in the delivery network; the attacker may also want to drop some ACKs. A secure feedback service should be able to detect and locate such attacks.

3 The Basic Scheme

Our secure feedback scheme has four main phases: aggregation tree construction, ACK aggregation and audit, aggregate verification and fault localization.

Aggregation tree construction. In our scheme, ACK aggregation is performed over an aggregation tree rooted at the BS. An intermediate node in the aggregation tree acts as an aggregator which aggregate ACKs from its child nodes. The aggregation tree construction process is a process of topology discovery. It determines which nodes are aggregators and which are not. The construction is performed once in the system initialization stage and it can be performed again whenever needed to reflect any network changes caused by mobility of certain nodes, or to the addition or deletion of sensor nodes.

ACK aggregation and audit. After the authentication tree is constructed, a node knows whether it is an aggregator or not. If a node is an aggregator, it has the information about its child nodes. Once the ACKs from its child nodes arrive, an aggregator aggregates them with the ACK of its own using an aggregation function ($Agg(\cdot)$) and then passes the aggregate value to its parent node in the tree. At the same time, the aggregator stores audit information for this operation. The audit information will be used in the fault localization stage to prove that the aggregator does perform its aggregation task honestly and to help the BS to locate errors in lower levels.

Aggregate verification. Upon receiving an aggregate, the BS, knowing the topology of the aggregation tree, reconstructs the aggregation tree. It then compares the computed aggregate with the one it received. If the two values are equal, the verification is successful and a successful verification shows that all the nodes acknowledged and all the ACKs arrived at the BS. Otherwise, the verification fails and the BS conducts a fault localization process to isolate nodes who fails in acknowledgement.

Fault localization. When an aggregate fails in the verification process, the BS asks the aggregator who generates the value for its audit information. The audit information

allows the BS to check whether this aggregator does its aggregation task honestly. The audit information also helps the BS to find which components of the aggregate are not correct. Then the BS asks the nodes responsible for these incorrect components in the lower level of the authentication tree for audit information. This process repeats until the leaf level of the authentication tree is reached. In each round, the BS narrows down its localization search one level down toward the leaf nodes in the aggregation tree.

3.1 Aggregation Tree Construction

We use the method described in TaG [13] to construct an aggregation tree. In TaG, the BS broadcasts a message asking sensors to organize into a routing tree. It specifies its own ID and its level (or distance from the root, in this case, zero) in that message. Any sensor node without an assigned level that hears this message assigns its own level to be the level in the message plus one. It also chooses the sender of the message as its parent, through which it will route messages to the root. Each of these sensor nodes then rebroadcasts the construction message, inserting their own IDs and levels. The construction message floods down the tree in this fashion, with each node rebroadcasting the message until all nodes have been assigned a level and a parent. The BS can initiate the construction process periodically so that to keep the latest network topology. To maintain stability in the network, parents are retained unless a child does not hear from them for some long period of time, at which point it selects a new parent using this same process. In Tag, sensors in adjacent area are implicitly grouped together through the construction process of the aggregation tree.

An example aggregation tree constructed using the method described above is shown in Figure 1. The tree is rooted at the BS. Intermediate nodes in the tree, such as nodes A , B , C , or E , act as aggregators. The tree construction process divides sensor nodes into groups and each group forms a sub-tree of the authentication tree. A group consists of sensor nodes which are geographically neighbored to each other. The root of the sub-tree performs the final aggregation operation and reports the result to the BS directly. For example, node A , M and N are the roots of such sub-trees and they report to the BS directly.

3.2 ACK Aggregation and Audit

Once the structure of the aggregation tree is known, ACKs can be aggregated when they are routed along the aggregation tree towards the root. Now we need to choose a proper aggregation function which an aggregator can use to combine ACKs coming from its child nodes.

Exclusive-Or (XOR) has been used in [14, 1] as the aggregation function to aggregate ACKs from sensors. An aggregator, upon receiving the ACKs from its child nodes, computes the XOR of all these ACKs with its own ACK. If all nodes acknowledged, the final aggregate value arrived at the BS is in the form of

$$ACK_1 \oplus ACK_2 \oplus \dots \oplus ACK_n$$

The XOR aggregation scheme is straightforward and independent on the structure of the aggregation tree. However, the security of the scheme is unclear as we do not know

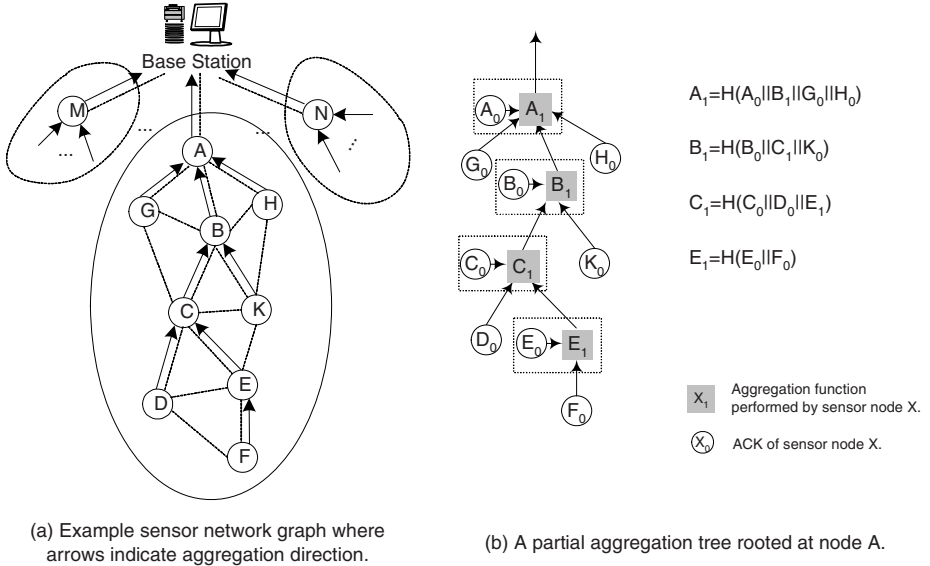


Fig. 1. Construction of an ACK aggregation tree

how strong the weak collision resistance of Exclusive-Or is. Weak collision resistance of Exclusive-Or in this setting can be defined as:

Given n messages $\{m_1, \dots, m_n\}$, the probability of finding n' messages $\{m'_1, \dots, m'_{n'}\}$ such that $m_1 \oplus \dots \oplus m_n = m'_1 \oplus \dots \oplus m'_{n'}$, $\{m_1, \dots, m_n\} \neq \{m'_1, \dots, m'_{n'}\}$ and $n' \leq n$.

We do not know how to calculate this probability. It is hard to evaluate the security strength of the XOR-based scheme. Instead we use a collision resistant hash function to aggregate ACKs. As in the XOR scheme, each sensor node generates its ACK in the form of a MAC: $ACK_S = MAC(K_S, N, S)$ where K_S is the secret key shared between the node S and the BS, N is the nonce or message ID unique to each broadcast message requesting acknowledgement and $MAC(\cdot)$ is any secure MAC scheme. The message a leaf node S sends to its parent is a tuple: (S, N, ACK_S) . An aggregator, upon receiving the ACKs from its child nodes, concatenates all the ACKs with its own in order of IDs and then calculates the hash value over the concatenation. The hash value of the concatenation is the aggregate value which the aggregator further forwards to its parent in the tree. The message an aggregator A sends to its parent is a tuple (A, N, AGG_A) where AGG_A denotes the aggregate value. Figure 1 (b) shows the aggregation function performed by aggregators A , B , C and E and the corresponding aggregate values calculated by them. Meanwhile, every aggregator stores all the ACKs used in the aggregation as audit information. That is, A stores B_1 , G_0 , H_0 and its own ACK A_0 ; B stores B_0 , C_1 and K_0 ; C stores C_0 , D_0 and E_1 ; E stores E_0 and F_0 .

We can view the process of ACK aggregation as constructing a hash tree distributively among all sensor nodes. To verify an aggregate, the BS reconstructs the sub-tree

by using the topology information and the secret keys shared with each sensor node in the group and compares its own calculated root value of the sub-tree with the aggregate value it received.

3.3 Fault Localization

When the verification of an aggregate failed, some nodes in the group do not acknowledge normally. BS initiates an iterative localization process among group members to locate nodes in problem. The localization begins with the root of a sub-tree who generates the final aggregate. The BS asks the root of the sub-tree for audit information which contains all the components in the final aggregate. The BS first checks whether the aggregator behaved honestly. It hashes the components extracted from the audit information and compares the hash result with the final aggregate it received. If the root is honest in aggregation, the two values will match. Next the BS checks whether a component matches the value it locally computed. An unmatched value tells the BS the fact that some nodes, who do not acknowledge normally, are in the sub-tree where this component value is the root value. Hence the BS will ask the node who generates the component for its audit information. By checking the audit information of an aggregator, the BS will narrow its search one level below this aggregator. That is, a test on the audit information of a level l node will let the BS locate some level $l + 1$ subgroups. The BS repeats this localization process one level further down to the leaf level until it reaches the leaf level.

We use an example to illustrate the concept of this iterative localization process. Suppose node F is under DoM attack. Its parent E either did not receive anything from it or received a forgery from the attacker. This error propagates in the aggregate values of E_1 , B_1 and A_1 . When the BS finds that A_1 does not match A_1^c which is locally computed by the BS (we use superscript c to denote that a value is a locally computed value by the BS), it asks A for audit information on A_0 , B_1 , G_0 and H_0 . It computes the hash over these components: $A_1' = H(A_0||B_1||G_0||H_0)$. If $A_1' = A_1$, A performed aggregation honestly. Next the BS checks whether any component of A_1 matches the value it locally computed: $A_0 \stackrel{?}{=} A_0^c$, $B_1 \stackrel{?}{=} B_1^c$, $G_0 \stackrel{?}{=} G_0^c$, and $H_0 \stackrel{?}{=} H_0^c$. The value B_1 will not match B_1^c in this case. Then the BS asks B to send its audit information on B_0 , C_1 and K_0 . It computes $B_1' = H(B_0||C_1||K_0)$ and compares B_1' with B_1 it received from A to check whether A indeed aggregated ACKs from its child nodes. By inquiring on B 's audit information, the BS finds that C_1 does not match C_1^c and asks C for audit information. Finally the BS finds that no ACK is from node F or $F_0 \neq F_0^c$ (F_0 is a forgery). That is, after four rounds of investigation the BS locates the error at sensor node F .

Security. An attacker is unable to make an individual ACK forgery without knowing the secret key because the underlying MAC scheme is unforgeable. An attacker is unable to make an aggregate forgery if at least one node is not compromised by the attacker. Otherwise we can either break the collision-resistance of the underlying hash function or the unforgeability of the MAC scheme.

In our scheme, an aggregator's role is just computing an aggregation operation. An aggregator is not necessarily to be trusted more by the BS than any other leaf nodes. An aggregator can not inject a forged ACK, modify an ACK or drop an ACK from its child nodes since any of these operations can be detected by the BS in the fault localization process. An aggregator can not provide irresponsible audit information to the BS as the BS will check its honesty in the localization process.

4 Improvements

4.1 Reducing Localization Delay

To locate an error originated from a leaf node in level l , the BS needs to perform l -rounds investigation process to locate the error. The height of the aggregation tree h determines the maximal delay in the localization process. For an aggregation tree constructed with TaG, the height of the aggregation tree depends on the node density and the total number of nodes n in the network. In a two-dimensional deployment area with a constant node density, the best bound on the diameter of the network is $O(\sqrt{n})$ if the network is regularly shaped. In irregular topologies the diameter of the network may be $\Omega(n)$. As the aggregation tree constructed by TaG may be arbitrarily unbalanced, the performance of the investigation process (in number of rounds) to locate errors in different levels varies dramatically. For example, to locate an error originated from F , the BS needs to perform four-rounds investigation process; on the other hand, to locate an error originated from H , the BS only needs to perform one round investigation process.

This motivates us to construct a more balanced aggregation tree to have localization delay bounded to $O(\log n)$. We use the *delayed aggregation* idea described in [1] to construct a balanced aggregation tree. In the delayed aggregation approach, an aggregator only computes the aggregate of *some* (not *all* as in our basic scheme) of its child ACKs and passes the other ones to its parent for aggregation. Child nodes whose ACKs are not aggregated and passed to its parent's parent are moved one level up towards the root. Hence, delayed aggregation helps reducing the height of the aggregation tree. It trades off increased communication during aggregation phase in return for a more balanced aggregation tree with a height of $O(\log_d n)$ where d is the degree of a node in the tree, and hence a better performance in the number of rounds in the fault localization phase.

Now we describe the algorithm for producing balanced aggregation trees with node degree of d . Our algorithm extends the algorithm which is used to construct a balanced binary tree [1]. We use the same strategy to construct a balanced d -ary aggregation tree: an aggregation operation is performed if and only if it results in a complete, d -ary aggregation tree. We assume each internal node keeps a small, fixed size list of neighborhood to maintain network topology.

We define a d -ary *forest* as a set of d -ary complete trees such that no d trees have the same height: $\{tree_1, tree_2, \dots\}$. A tree in the forest is represented by its root node and the number of leaf nodes in the tree: $tree = (ID, count)$. A leaf node V in the aggregation tree generates a forest with a single node tree in the form $\{(V, 1)\}$ and sends it to its parent. An internal node S generates its own forest in the similar way. In

addition, S also receives forests from its children. S combines all the forests to form a new forest as follows. Suppose S wishes to combine q forests $\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_q$. Let $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2 \cup \dots \cup \mathcal{F}_q$. If there are less than d trees in \mathcal{F} , S simply passes \mathcal{F} to its parent and no aggregation operation is performed by S . The height of a tree in \mathcal{F} can be determined by inspecting the *count* field of the tree. Let h be the smallest height such that more than d trees in \mathcal{F} has height h . S picks up d trees T_1, T_2, \dots, T_d of height h , and merge them into a tree of height $h + 1$. The rule for S to pick up d trees of the same height to form a new tree is it always picks up trees with root nodes closer to itself. S obtains the adjacent nodes information from its neighborhood list. This process repeats until no two trees are the same height in \mathcal{F} . Then S forwards the new forest to its parent. Aggregation happens when the combination of trees happens.

Figure 2 shows an example of the process to construct a balanced 3-ary aggregation tree rooted at node A . When E receives a forest $\mathcal{F}_F = \{(F_0, 1)\}$ from its child F , as only two trees are in the combined forest $\mathcal{F}_E = \{(E_0, 1), (F_0, 1)\}$, E simply forwards \mathcal{F}_E to its parent C . C picks three trees, $(C_0, 1)$, $(D_0, 1)$ and $(E_0, 1)$, of height 0 and whose roots are closer to it to form a tree of height 1: $(C_1, 3)$. C then forwards the forest $\mathcal{F}_C = \{(C_1, 3), (F_0, 1)\}$ to its parent B . B forms a new tree $(B_1, 3)$ of height 1 with trees $(B_0, 1)$, $(F_0, 1)$ and $(K_0, 1)$ and sends the forest $\mathcal{F}_B = \{(B_1, 3), (C_1, 3)\}$ to A . A combines trees $(A_0, 1)$, $(G_0, 1)$ and $(H_0, 1)$ to forms a tree $(A_1, 3)$ of height 1. A further combines trees $(A_1, 3)$, $(B_1, 3)$ and $(C_1, 3)$ to form a tree $(A_2, 9)$ of height 2.

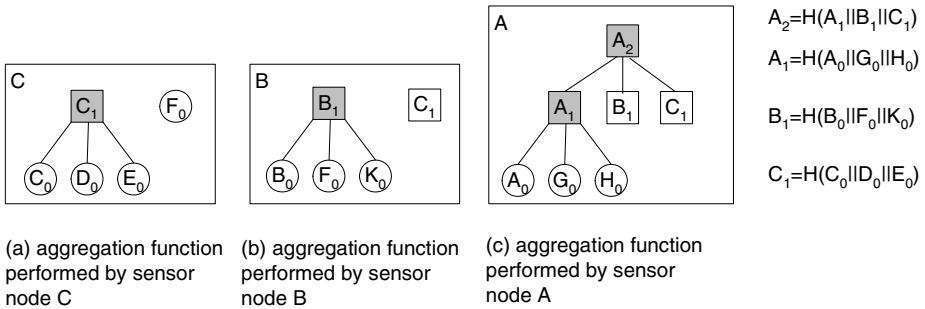


Fig. 2. Improved construction of balanced aggregation tree

In the new balanced aggregation tree rooted at A of height 2, E is no longer an aggregator. The three aggregators now are A , B and C . A performs two aggregation operations. It first aggregates ACKs from G , H with its own ACK to get aggregate value A_1 . It then further aggregates A_1 with B_1 and C_1 . Aggregate values generated by each aggregator are listed in Figure 2. Note in the new balanced aggregation tree, F is now in level 2 and its parent is B . Using the same example shown in Section 3.3, the BS now only needs to perform two rounds of the investigation process to locate an error originated from F .

4.2 Reducing Space for Storing Audit Information

An aggregator needs to store at least d ACKs as audit information (in Figure 2, A needs to store $2d$ ACKs as it performs aggregation twice). Instead of storing these ACKs separately, we can use a space-efficient data structure, Bloom filter, to store those ACKs.

The Bloom filter is conceived by Burton H. Bloom in 1970 [3]. It is a probabilistic data structure for testing membership of a set [16]. A Bloom filter for representing a set $S = \{s_1, s_2, \dots, s_n\}$ of n elements is described as an array of m bits, initially all set to 0. A Bloom filter uses k independent hash functions h_1, h_2, \dots, h_k which map each a key value to one of the m positions uniformly. For each element $s \in S$, the bits $h_i(s)$ are set to 1 for all $1 \leq i \leq k$. A location can be set to 1 multiple times, but only the first change has an effect. To check if an item x is in S , we check whether all $h_i(x)$ are set to 1. If not, then clearly x is not a member of S . If all $h_i(x)$ are set to 1, we assume that x is in S . A Bloom filter may yield a false positive (but no false negative error), where it suggests that an element x is in S even though it is not. The probability of a false positive for an element not in the set, or the false positive rate p , is calculated as:

$$p = (1 - (1 - \frac{1}{m})^{kn})^k \approx (1 - e^{-kn/m})^k$$

For a given m and n , the value of k that minimizes the probability is

$$k = \frac{m}{n} \ln 2 \approx 0.7 \frac{m}{n}$$

which gives a probability of

$$p = (2^{-\ln 2})^{m/n} \approx 0.62^{m/n}$$

Bloom filters have a strong space advantage over other data structures for representing sets. A Bloom filter with 1% error and an optimal value of k , on the other hand, requires only about 9.6 bits per element - regardless of the size of the elements. This advantage comes partly from its compactness and partly from its probabilistic nature. If a 1% false positive rate seems too high, each time we add about 4.8 bits per element we decrease it by ten times.

We modify our fault localization process to accommodate the use of Bloom filters to store audit information. For each aggregation operation, an aggregator computes a Bloom filter with all its input ACKs as members. When the Bloom filter of node S at level l arrives at the BS, the BS tests the Bloom filter with its locally computed ACKs from S 's child nodes at level $l+1$ (inputs to S 's aggregation function). If the test shows that some ACKs are not in the Bloom filter, it tells the BS that some errors are in the lower groups with these level $l+1$ nodes as roots. In other words, a test on an level l Bloom filter let the BS narrow its search to some level $l+1$ subgroups. The BS repeats the investigation process one level further down to the leaf level until it reaches the leaf level. A Bloom filter has false positives. A false positive happens when the BS cannot find any missed ACK value and thus it cannot locate a subgroup in a lower level to proceed the localization process. In this case, the BS simply asks all the child nodes of that aggregator to send their individual ACKs directly to the BS. We will analyze how the false positive affect the performance of the localization process in Section 5.

5 Analysis

We compare the communication overhead of our secure feedback service scheme *SFSS* with the naive scheme *No-Agg* where individual ACKs are sent to the BS directly without aggregation. To simplify the measurements, we envision a WSN, with numerous sensor nodes and only one base station, organized into a balanced d -ary aggregation tree of height h . The number of bits sent by individual nodes and the overall bandwidth in the WSN are measured over this tree.

5.1 Per Node Communication Cost

For the *No-Agg* scheme, a leaf node only needs to transmit its own acknowledge message and the message length is $|M| = |ID| + |Nonce| + |ACK|$ in bits. All internal nodes need to forward the packets sent to them by their children, and the number of packets received grows exponentially as we move higher in the tree or closer to the root.

The number of bits a node at level l needs to transfer is calculated as $\frac{d^{h-l+1}-1}{d^l-1} * |M|$.

For our *SFSS* scheme, in the ACK aggregation phase, each node forwards the same number of bits to its parent; in the fault localization phase, an aggregator involved in the investigation process needs to send its Bloom filter containing its audit information to the BS. Thus for leaf nodes and aggregators not involved in the investigation process, the number of bits they need to send is $|M|$; an aggregator involved in the investigation process needs to send $|M| + |BF|$ bits where $|BF|$ denotes the length of a Bloom filter in bits. Considering the situation when a false positive happens to an aggregator's Bloom filter, ACKs from its child nodes are required to be sent to the BS directly and this imposes $(d-1) * |M|$ communication load for level l nodes. Suppose there are n_l^{inv} nodes in level l involved in the investigation process, the average communication cost per node at level l is calculated as $(1 + \frac{n_l^{inv} * p * d}{d^l}) * |M| + \frac{n_l^{inv}}{d^l} * |BF|$. If we ignore the false positive of the Bloom filter such that $p = 0$, the per node cost is $|M| + \frac{n_l^{inv}}{d^l} * |ACK|$. If we further assume that all nodes acknowledged correctly such that $n_l^{inv} = 0$, we get the per node cost in this ideal situation as $|M|$.

From the analysis above, we know for the *No-Agg* approach there is a widely differing data communication load amongst sensors at different levels. The nodes closer to the sink die sooner as they have to send significantly larger amounts (d times) of data than their descendants. Instead ACK aggregation in *SFSS* mitigates the burden of higher level nodes on forwarding packets so that all the nodes roughly have the same transmission load. A level 1 nodes in *SFSS* only have a transmission load approximately d^h times less than that of a level 1 node in *No-Agg*.

5.2 Overall Communication Cost and Bandwidth Gain

The overall communication cost in the WSN is computed as the sum of the communication cost at each level in the tree. For the *No-Agg* scheme, the overall communication cost is calculated as: $C_{No-Agg} = \sum_{l=1}^h d^l * \frac{d^{h-l+1}-1}{d^l-1} * |PKT|$. For the *SFSS* scheme, the overall communication cost is calculated as: $C_{SFSS} = \sum_{l=1}^h (1 + \frac{n_l * p * d}{d^l}) * |M| + \frac{n_l}{d^l} * |BF|$. The bandwidth gain of *SFSS* over *No-Agg* is defined as $\frac{C_{No-Agg}}{C_{SFSS}}$.

Now we use concrete examples to show the bandwidth gain of *SFSS* over *No-Agg*. We consider a balanced aggregation tree of degree of 10 and height of 3, that is $d = 10$ and $h = 3$. Based on the same tree topology, we calculate the bandwidth gain with three different Bloom filter false positive rates: $p = 1\%$, 0.1% , 0.01% . When $p = 1\%$, the length of a bloom filter $|BF|$ containing 10 elements is 96-bits; when $p = 0.1\%$, $|BF| = 144$ -bits; when $p = 0.01\%$, $|BF| = 192$ -bits.

We consider three scenarios: when (1) all the nodes reply; (2) 90% of the nodes reply and (3) 70% of the nodes reply. The bandwidth cost in *Agg+Invt* is dependent on the distribution of nodes who fail in reply. Therefore we consider two extreme cases: (1) the worst case when errors occur in the maximum number groups; (2) the best case when errors occur in the minimum number of groups. This translates that in the 90% case, $\text{MAX}(n_1^{invt})=10$, $\text{MAX}(n_2^{invt})=100$ and $\text{MIN}(n_1^{invt})=1$, $\text{MIN}(n_2^{invt})=1$; in the 70% case, $\text{MAX}(n_1^{invt})=10$, $\text{MAX}(n_2^{invt})=100$ and $\text{MIN}(n_1^{invt})=1$, $\text{MIN}(n_2^{invt})=3$. We use 90%-MAX, 90%-MIN, 70%-MAX, and 70%-MIN to denote these different cases. Bandwidth gain with error distributions different from these two extreme distributions is in between the bandwidth gains of these two extremes. The calculation results are listed in Table 1.

Table 1. Bandwidth Gain (*Agg+Invt* vs. *No-Agg*.)

d	fp	m	100%	90%-MIN	90%-MAX	70%-MIN	70%-MAX
10	1%	96	2.7	2.43	2.14	1.88	1.66
10	0.1%	144	2.7	2.42	2.04	1.88	1.58
10	0.01%	192	2.7	2.42	1.93	1.87	1.5

From Table 1, we see *SFSS* has a good bandwidth gain over the naive scheme *No-Agg*. In the MAX distribution case, although decreasing false positive rate of the bloom filter decreases the probability for nodes to re-send their ACKs directly to the BS when their parent's Bloom filter has a false positive, it increases the length of all the bloom filters and hence leads to a decrease of the total bandwidth gain. In the MIN distribution case, false positive rate does not affect the bandwidth gain too much as in this case investigation traffic only contributes a very small percentage of the total bandwidth cost.

The results shown in this section are very encouraging since they confirm that aggregation is a useful technique for reducing the total bandwidth usage and therefore extend the overall lifetime of the network.

6 Conclusion

In this paper, we proposed a secure feedback scheme to provide secure feedback service in some sensor applications. In our basic scheme, ACKs are aggregated when they travel along the aggregation tree rooted at the BS. Each aggregator stores audit information which allows the BS to locate errors in the network. Improvements are made to reduce fault localization delay and storage overhead for audit information. Performance analysis showed that our scheme achieves good bandwidth gain over the naive scheme and enable a longer life of the WSN.

References

1. H. Chan, A. Perrig, and D. Song. "Secure hierarchical in-network aggregation in sensor networks". *ACM CCS'06*, Nov. 2006.
2. K. Barr, and K. Asanovic. "Energy aware lossless data compression". In *Proc. of MobiSys'03*. San Francisco, CA, May 2003.
3. B. Bloom. "Space/time tradeoffs in hash coding with allowable errors". *Communication of the ACM*, 13(7):422-426, July 1970.
4. A. Boldyreva. "Efficient threshold signature, multisignature and blind signature scheme based on the gap-Diffe-Hellman-group signature scheme". In *Proc. of PKC 2003*, LNCS 2567, pp. 31-46, Springer-Verlag, 2003.
5. D. Boneh, C. Gentry, B. Lynn, and H. Shacham. "Aggregate and verifiably encrypted signatures from bilinear maps". In *Proc. of Eurocrypt 2003*, LNCS 2656:416-432, May 2003.
6. C. Castelluccia, S. Jarecki, J. Kim, and G. Tsudik. "A robust multisignature scheme with application to acknowledgement aggregation". In *Security in Communication Networks '04*, 2004.
7. C. Castelluccia, E. Mykletun, and G. Tsudik. "Efficient aggregation of encrypted data in wireless networks". In *Mobile and Ubiquitous Systems: Networking and Services MobiQuitous 2005*. July 2005.
8. L. Hu, and D. Evans. "Secure aggregation for wireless networks". In *Workshop on Security and Assurance in Ad Hoc Networks*, 2003.
9. W. Lin, S. M. Chang, S. P. Shieh. "An efficient broadcast authentication scheme in wireless sensor networks". *ASIACCS 2006*.
10. C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. "Impact of network density on data aggregation in wireless sensor networks". In *ICDCS'02*, pp. 457-458. 2002.
11. S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters. "Sequential aggregate signatures and multisignatures without random oracles". In *Proc. of Eurocrypt 2006*, May 2006.
12. A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham. "Sequential aggregate signatures from trapdoor permutations". In *Proc. of Eurocrypt 2004*, LNCS 3027:245-254, Nov. 2001.
13. S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. "TAG: a tiny aggregation services for ad-hoc sensor networks". *SIGOPS Oper. Syst. Rev.* 36(SI):131-146, 2002.
14. J. McCune, E. Shi, A. Perrig, and M. Reiter. "Detection of Denial-of-Message Attacks on Sensor Network Broadcasts". *IEEE Symposium on Security and Privacy*, 2005.
15. S. Micali, K. Ohta, and L. Reyzin. "Accountable-subgroup multisignatures". In *Proc. of CCS 2001*, pp. 234-54, 2001.
16. M. Mitzenmacher. "Compressed bloom filters". *IEEE/ACM Trans. on Networks*. pp 613-620. Oct 2002.
17. T. Okamoto. "A digital multisignature scheme using bijective public-key cryptosystems". *ACM Trans. Computer Systems*, 6(4):432-441, 1998.
18. S. Park, R. Vedantham, R. Sivakumar, and I. Akyildiz. "A scalable approach for reliable downstream data delivery in wireless sensor networks". *MobiHoc 2004*. May 24-26, 2004.
19. A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler. "SPINS: Security protocols for sensor networks". *Wirel. Netw.*, 8(5):521-534, 2002.
20. D. Wagner. "Resilient aggregation in sensor networks". In *Workshops on Security of Ad Hoc and Sensor Networks*. 2004.
21. Y. Yang, X. Wang, S. Zhu, and G. Cao. "SDAP: a secure hop-by-hop data aggregation protocol for sensor networks". In *ACM MOBIHOC'06*. May 2006.