Statistical En-route Filtering of Injected False Data in Sensor Networks

Fan Ye[1], Haiyun Luo[2], Songwu Lu[3], and Lixia Zhang[3]

[1]IBM T.J. Watson Research, 19 Skyline Drive, Hawthorne, NY 10532, fanye@us.ibm.com
[2]Dept. of Computer Science, University of Illinois, Urbana, IL 61801, haiyun@cs.uiuc.edu
[3]Computer Science Department, UCLA, CA 90095, {slu,lixia}@cs.ucla.edu

***Abstract*— In a large-scale sensor network individual sensors are subject to security compromises. A compromised node can be used to inject bogus sensing reports. If undetected, these bogus reports would be forwarded to the data collection point (i.e. the sink). Such attacks by compromised nodes can result in not only false alarms but also the depletion of the finite amount of energy in a battery powered network. In this paper we present a Statistical En-route Filtering mechanism (SEF) to detect and drop false reports during the forwarding process. Assuming that the same event can be detected by multiple sensors, in SEF each of the detecting sensors generates a keyed message authentication code (MAC) and multiple MACs are attached to the event report. As the report is forwarded, each node along the way verifies the correctness of the MACs probabilistically and drops those with invalid MACs. SEF exploits the network scale to filter out false reports through collective decision-making by multiple detecting nodes and collective false detection by multiple forwarding nodes. We have evaluated SEF's feasibility and performance through analysis, simulation, and implementation. Our results show that SEF can be implemented efficiently in sensor nodes as small as Mica2. It can drop up to 70% of bogus reports injected by a compromised node within 5 hops, and reduce energy consumption by 65% or more in many cases.**

***Index Terms*— false data injection, compromised nodes, en-route filtering**

I. Introduction

Wireless sensor networks are expected to interact with the physical world at an unprecedented level to enable various new applications. However a large scale sensor network may be deployed in a potentially adverse or even hostile environment and potential threats can range from accidental node failures to intentional tampering. Due to their relatively small sizes and unattended operations, sensor nodes have a high risk of being captured and compromised. False sensing reports can be injected through compromised nodes, which can lead to not only false alarms but also the depletion of limited energy resource in a battery powered network. Although several recent research efforts [1], [2], [3] have proposed mechanisms to enable node and message authentication in sensor networks, those proposed solutions can only prevent false reports injection by outside attackers. They are rendered ineffective when any single node is compromised.

To combat false reports injected by compromised nodes one must have means to detect such false reports. However developing such a detection mechanism represents a great research challenge. On the one hand, the computation and storage constraints of small sensor nodes make asymmetric cryptography based mechanisms, such as the one described in [4], infeasible. On the other hand, straightforward usage of symmetric keys is infeasible because once a node is compromised, all the shared security information stored in that node can be used by an attacker. The compromised node can successfully authenticate bogus reports to a neighbor, which has no way to differentiate such false reports from legitimate ones. Moreover, the effectiveness of detection schemes based on application semantics may be limited when the attacker knows application semantics and injects only bogus reports that do not violate the semantics, and their potentials remain to be further explored and thoroughly evaluated. Finally, effective DDoS packet filtering solutions developed for the Internet that utilize structural properties of the infrastructure, such as the one proposed in [5], are not applicable to ad-hoc deployed, self-organized sensor networks with a flat topology.

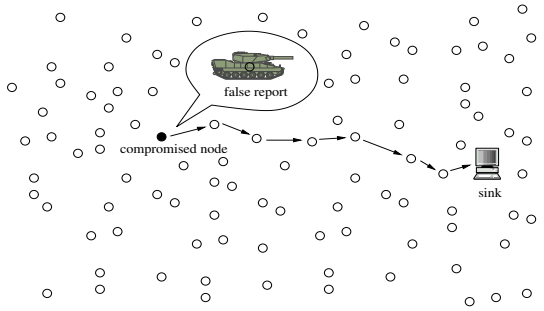

Fig. 1. A compromised node injects false reports of non-existent "tanks " events. Such bogus reports can mislead reactions, delay or block legitimate reports by occupying the communication channel, and drain out network energy.

In this paper we present a Statistical En-route Filtering (SEF) mechanism. SEF exploits the sheer scale and dense deployment of large sensor networks. To prevent any single compromised node from breaking down the entire system, SEF carefully limits the amount of security information assigned to each node, and relies on the collective decisions of multiple sensors for false report detection. When a sensing target (henceforth called either "stimulus" or "event") appears in the field, multiple surrounding sensors collectively generate a

This paper is an extension of a shorter version appeared at INFOCOM'04. Manuscript received November, 2003, revised July, 2004.

legitimate report and endorse it by attaching to it their message authentication codes (MACs); a report with an inadequate number of MACs will be dropped. As a report is forwarded through multiple hops towards the sink, each forwarding node verifies the correctness of the MACs carried in the report with certain probability and drops the report if an incorrect MAC is detected. The probability of detecting incorrect MACs increases with the number of hops the report travels. Due to the statistical nature of the detection mechanism, a few bogus reports with incorrect MACs may escape en-route filtering and reach the sink. However, the sink can further verify the correctness of each MAC and reject false reports.

The contribution of this paper is two-fold. First, we propose a key assignment method designed for en-route detection of false reports in the presence of compromised nodes. Second, we develop mechanisms for collective data report endorsement, en-route report filtering and sink verification. To our best knowledge this is the first effort that addresses false report detection problems *in the presence of compromised sensor nodes*. We have evaluated our design through analysis, implementation and simulations. Our results show that SEF is able to drop up to 70% of bogus reports injected by a compromised node within 5 hops, and up to 90% within 10 hops along the forwarding paths. The en-route filtering adds computation complexity of less than 1ms time and about 75 μJ energy on each forwarding node, while reducing total energy consumption by 65% or more in many scenarios.

The rest of the paper is organized as follows. Section II and III present the model and design of SEF. Section IV discusses the parameter setting and analyses the effectiveness and energy savings achieved by SEF through measurements from implementation, and evaluates the design through simulations. A number of practical issues and future work are discussed in Section V. Section VI compares SEF with the related work and Section VII concludes the paper.

II. System Models and Assumptions

A. Sensor Network Model

We consider a sensor network composed of a large number of small sensor nodes. We further assumes that the sensor nodes are deployed in high density, so that a stimulus (e.g., a tank) can be detected by multiple sensors. Each of the detecting sensors reports its sensed signal density and one of them is elected as the Center-of-Stimulus (CoS) node. The CoS collects and summarizes all the received detection results, and produces a synthesized report on behalf of the group. The report is then forwarded toward the sink, potentially traversing a large number of hops (e.g. tens or more). The sink is a data collection center with sufficient computation and storage capabilities, and it may also implement advanced security solutions to protect itself.

Due to cost constraints we assume that each sensor node is *not* equipped with tamper-resistant hardware[1]. However, dense deployment enables cross-verification of a reported event among multiple sensors even in the presence of one or more compromised nodes. SEF design harnesses the advantage of large-scale. Rather than relying on a small number of powerful and expensive sensors, SEF utilizes large numbers of small sensors for reliable sensing and reporting.

[1]Tamper-resistant hardware can prevent the exposure of stored secrets[6] even when a node is captured by the attacker.

B. Threat Model

We assume that the attacker may know the basic approaches of the deployed security mechanisms, and may be able to either compromise a node through the radio communication channel, or even physically capture a node to obtain the security information installed in the node. However, we assume that attacks cannot subvert the data collection unit, i.e., the sink, because the protection at the sink is powerful enough to defeat such subversion efforts. Once compromised, a node can be used to inject false reports into the sensor network. Node and message authentication mechanisms [1], [2], [3] prevent naive impersonation of a sensor node. However, they cannot block false injection of sensing reports by compromised nodes.

Besides false data injection, a compromised sensor node can launch various other attacks. It can stall the generation of reports for real events, block legitimate reports from passing through it (which we call *false negative attacks*), or record and replay old reports, etc. As the first effort in tackling the threats from compromised components, this paper focuses on the detection of false event reports, which we call *false positives attacks*, injected by compromised nodes. We plan to address other attacks in subsequent efforts.

III. Statistical En-route Filtering

In this section, we present the SEF design. Section III-A gives an overview of the design, followed by a more detailed description of the three major components of SEF, key assignment and report generation, en-route filtering, and sink verification in Sections III-B, III-C, and III-D, respectively. Finally we discuss how to reduce overhead with two different techniques in Section III-E (with more detailed comparison in Section IV-B).

A. Overview

SEF aims at achieving the following goals:

- *Early detection of false data reports*: by detecting false reports the user can avoid responding to fabricated events. Although the detection can be done after the data reports arrive at the sink, en-route early detection and dropping of false reports can conserve energy and bandwidth resources of sensor nodes along data forwarding paths.
- *Low computation and communication overhead*: Given the resource constraints of low-end sensor nodes, we rule out solutions based on computation-intensive asymmetric cryptography[2] and only use efficient one-way functions.

SEF consists of three components that work in concert to detect and filter out forged messages: (1) each legitimate report

[2]It has been reported that encryption/decryption operations based on asymmetric keys consume two to three magnitudes more energy than symmetric ones, sometimes signing a bit is even more expensive than transmitting a bit [7].

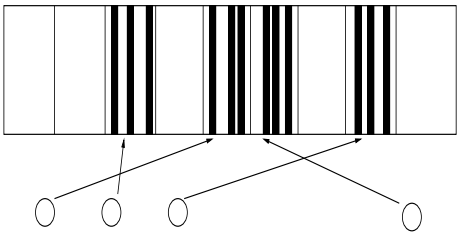

Fig. 2. An example of a global key pool with $n = 9$ partitions and 4 nodes, each of which has $k = 3$ keys randomly selected from one partition. In a real system, k, n can be much larger.

carries multiple MACs generated by different nodes that detect the same stimulus, (2) intermediate forwarding nodes detect incorrect MACs and filter out false reports *en-route*, and (3) the sink verifies the correctness of each MAC and eliminates remaining false reports that elude en-route filtering.

In SEF, the sink maintains a global key pool. Each sensor stores a small number of keys that are drawn in a randomized fashion from the global key pool before deployment. Whenever a stimulus appears in the sensor field, multiple surrounding nodes can detect the event and a CoS node is elected to generate the event report. Each detecting sensor endorses the report by producing a keyed MAC using one of its stored keys. The CoS node collects the MACs and attaches them to the report. This set of multiple MACs acts as the proof that a report is legitimate. A report with insufficient number of MACs will not be forwarded.

The key assignment ensures that each node can generate only *partial* proof for a report. Only by the joint efforts of multiple detecting nodes can the complete proof be produced. A single compromised node has to forge MACs to assemble a seemingly complete proof in order for the forged data report to be forwarded. Because nodes share common keys with certain probabilities, when the report with forged MACs is forwarded by intermediate nodes, the nodes can verify the correctness of the MACs probabilistically, thus detecting and dropping false ones en-route.

The sink serves as the final goal-keeper for the system. When it receives event reports, the sink can verify all the MACs carried in the report because it has complete knowledge of the global key pool. False reports with incorrect MACs that sneak through en-route filtering will then be detected.

Several questions in the above design must be answered: (a) How should the keys be assigned to nodes to prevent a compromised node from forging the complete proof while enabling verification by intermediate forwarding nodes? (b) How are false reports detected and filtered out en-route by forwarding sensors? (c) What procedures does the sink follow to detect any remaining forged reports? And (d) How can the size of the multiple MACs carried in a report be kept minimal to reduce the overhead? The rest of this section addresses each of these four questions in order.

B. Key Assignment and Report Generation

There is a pre-generated global pool of N keys $\{K_i, 0 \leq i \leq N-1\}$, divided into n non-overlapping partitions $\{N_i, 0 \leq i \leq n-1\}$. Each partition has m keys (i.e., $N = n \times m$), and each key has a unique key index. A simple way to partition the global key pool is as follows:

$$N_i = \{K_j | im \leq j \leq (i+1)m - 1\}.$$

Before a sensor node is deployed, the user randomly selects one of the n partitions, and randomly chooses $k (k < m)$ keys from this partition to be loaded into the sensor node, together with the associated key indices (see Figure 2 for an example).

When a stimulus appears, all surrounding nodes that detect the signal will prepare an event report in the form of $\{L_E, t, E\}$, where L_E is the location of the event, t is the time of detection and E is the type of event[3]. Similar to [8], each detecting node sets a random timer, upon the timer expiration it broadcasts its values of $\{L_E, t, E\}$. If another node finds the difference between the broadcast values and what it observes is within some pre-defined error range, it accepts them and cancels its own timer. Otherwise it broadcasts its own observed values on expiration of its timer. The node whose broadcast values are accepted by others becomes the Center of Stimulus (CoS) node.

When real events occur in the sensor field, the CoS election allows detecting nodes to generate MACs for the same report content in the absence of compromised nodes. The pre-defined error range is decided based on the sensing accuracy of nodes and the application's requirements to suppress duplicate data report generation. However, when no real event occurs, well-behaving sensors will not send out detection results and the associated MACs. Therefore, the compromised node(s) (in a worst-case scenario, even the CoS itself may be compromised) are forced to forge MACs in order to generate false reports, which can be subsequently detected by SEF. However, note that SEF does not address false negative attacks. A compromised node can stall the proper reporting of a real event by sending many incorrect values to block the communication channel. For applications that require more complex decision making where the report contains more than the target type and each sensing node has very different detecting result, a consensus needs to be reached before the report can be endorsed, which is beyond the scope of this paper.

After the election process finishes, a detecting node A randomly selects K_i, one of its k keys, and generates a MAC

$$M_i = \overline{MAC}(K_i, L_E||t||E), \tag{1}$$

where $||$ denotes stream concatenation and $\overline{MAC}(a, b)$ computes the MAC M_i of message b using key a. Many cryptographic one-way functions may serve this purpose [9]. The node then sends $\{i, M_i\}$, the key index and the MAC, to the CoS. The CoS collects all the $\{i, M_i\}$'s from detecting nodes and classify MACs based on the key partitions. We define MACs generated by keys of the same partition as one *category*. Suppose CoS collects T categories ($T \leq n$). From each category, the CoS randomly chooses one $\{i, M_i\}$ tuple and attaches it to the report. The final report sent out by the

[3]The report may also contain other information about the event. To simplify presentation, we only list the above three.

1) Check that T $\{i_j, M_{i_j}\}$ tuples exist in the packet; drop the packet otherwise.
2) Check the T key indices $\{i_j, 1 \leq j \leq T\}$ belong to T distinct partitions; drop the packet otherwise.
3) If it has one key $K \in \{K_{i_j}, 1 \leq j \leq T\}$, it computes $M = \overline{MAC}(K, L_E||t||E)$ as in Equation 1 and see if the corresponding M_{i_j} is the same as M. If so, it sends the packet to the next hop; otherwise the packet is dropped,
4) If it does not have any of the keys in $\{K_{i_j}, 1 \leq j \leq T\}$, sends the packet to the next hop.

Fig. 3. Operations in En-route Filtering

CoS to the sink looks like:

$$\{L_E, t, E, i_1, M_{i1}, i_2, M_{i2}, \cdots, i_T, M_{i_T}\}.$$

The choice of parameter T trades-off between detection power and overhead. The sink can set a system-wide value for T so that each report carries exactly T key indices of distinct partitions and T MACs. A report with less than T MACs or key indices, or more than one key index in the same partition, will not be forwarded. A larger T value makes forging reports more difficult at the cost of increased overhead. When more than T categories exist, the CoS can randomly choose T of them. In areas with sparse sensor deployment, there could be less than T categories. The node density should be high enough so that such cases rarely happen. More analysis on setting parameters will be given in Section IV.

C. En-route Filtering

As a result of the randomized key assignment, each forwarding node has certain probability to possess one of the keys that are used to generate the MACs in a data report, i.e., $\{K_{i_j}, 1 \leq j \leq T\}$, and verify the correctness of the corresponding MAC, i.e., $\{M_{i_j}, 1 \leq j \leq T\}$. A compromised node has keys from only one partition and can only generate MACs of one category. Since T MACs of distinct categories and T key indices of distinct partitions must be present in a legitimate report, the compromised node needs to forge the other $T-1$ key indices and corresponding MACs. This explains why the global key pool is partitioned: Had each node carried keys chosen from the entire pool, one compromised node can use T of its keys to generate multiple MACs, which would be indistinguishable from those generated by T nodes.

When a node receives a report, it first examines whether there are T key indices of distinct partitions and T MACs in the packet. Packets with less than T key indices, or less than T MACs, or more than one key index in the same partition are dropped. Then if the node has any of the T keys indicated by the key indices, it re-produces the MAC using its own key and compares the result with the corresponding MAC attached in the packet. The packet is dropped if the attached one differs from the reproduced. Only if they match exactly, or this node does not possess any of the T keys, the node passes the packet to the next hop. The pseudo code for en-route filtering operations is given in Figure 3.

If an attached MAC differs from the one locally produced by a forwarding node, it indicates that the report was not generated with the correct key. Such a MAC is considered forged, and the packet is dropped. Note that a forwarding node sends the report downstream if it does not have any of the T keys, because the report might be a legitimate one with MACs by keys not possessed by this node. This may cause a forged MAC to escape the screening of certain node. But the forged report will be detected and dropped with higher and higher probabilities as it travels more and more hops. The detection power of a single sensor is constrained, but the collective detection power grows as more nodes deliver the report. We will further analyze the performance in Section IV.

D. Sink Verification

When the sink receives a report, it can check the correctness of every M_{i_j} because it has all the keys. Any forged MAC that eludes the en-route filtering by chance will be caught. When the sink receives a report, it first examines whether the report carries T key indices of distinct partitions and T MACs similarly. It then re-computes each of the T MACs and compares them with the attached ones. If one mismatch happens, the packet is discarded.

Any forged MAC that passes en-route filtering can be detected by the sink. The sink serves as the final defense that catches false reports not filtered out by forwarding nodes. As long as a MAC is forged, the sink can detect and discard the report. Therefore, SEF can detect bogus reports forged by an attacker compromising keys in up to $T-1$ partitions. We further analyze SEF's detection power in Section IV.

E. Reducing the MAC Size

In addition to sensing data, each report carries T key indices and MACs, both of which increase the packet length and transmission energy consumption. Some sensor networks may also have stringent limit on the packet length due to hardware or software configuration (e.g., TinyOS[10] uses less than 36-byte packets), or potentially high error rates. In SEF MAC poses a main source of packet size increase. However we cannot reduce the MAC size by decreasing T, because a too small T value reduces SEF's false detection power. In the following, we present two techniques for reducing the overhead while retaining certain security protection strength. This raises an interesting question, which technique is better? We will compare their strength in Section IV-B and point out under what conditions which is better.

1) Shorter MACs: One method is to use MACs of shorter lengths. This reduces the overhead directly, at the cost of increasing the chance that an attacker can "guess" the MAC correctly. But it still requires multiple nodes to collectively endorse an event.

2) Bloom filters: We propose a second technique, Bloom filter, where we use a much shorter bit-string, instead of a list of MACs, to reduce packet size while retaining the false data detecting power.

Bloom Filter is a well-known data structure that can be used for efficient membership checking, i.e., given an element, find whether it is in a pre-defined set. A Bloom filter is made of

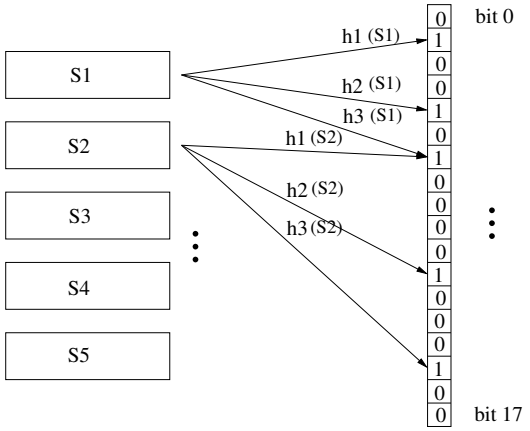

Fig. 4. A Bloom filter that represent $n = 5$ elements using a string of $m = 18$ bits and $k = 3$ hash functions, each maps any element to range $\{0, \cdots, 17\}$. Notice $h_3(s_1)$ and $h_1(s_2)$ both map to bit 6.

a set $S = \{s_1, s_2, \cdots, s_n\}$ using a string of m bits and k independent hash functions $h_1, h_2, \cdots, h_k$ [11]. Each h_i maps an item s uniformly to the range $\{0, 1, \cdots, m-1\}$, each of which corresponding to a bit in the m-bit string. (Note that we use n, m, k, h for different meanings in this section). The m-bit string is initially set to 0.

For each element $s \in S$, we hash it with all the k hash functions and obtain their values $h_i(s)(1 \leq i \leq k)$. The bits corresponding to these values are then set to 1 in the string. Note that more than one of the kn values may map to the same bit in the string (see Figure 4 for an example). To find whether an item $s' \in S$, bits $h_i(s')$ are checked. If all of them are 1, s' is considered to belong to S; s' is definitely not in S if at least one of them is 0.

Bloom filter may yield *false positives*, i.e., an element is not in S but its bits $h_i(s)$ are collectively marked by other elements in S. If the hash is uniformly random over the m values, the probability that a bit is 0 after all the n elements are hashed and their bits marked is $(1-\frac{1}{m})^{kn} \approx e^{-\frac{kn}{m}}$. Therefore, the probability for a false positive (i.e., the k bits of an element s are already marked) is $(1-(1-\frac{1}{m})^{kn})^k \approx (1-e^{-\frac{kn}{m}})^k$.

Certain changes are needed to use Bloom filter for SEF. The CoS applies k system-wide hash functions to map the T selected MACs (each with b bits) to a m-bit string $F = b_0 b_1 \cdots b_{m-1}$, where we have $m < bT$ to reduce packet size and $kT < m$ to retain en-route filtering capability. These k functions are known by every node and the sink. For each $b_l (0 \leq l \leq m-1)$, we have

$$b_l = \begin{cases} 1 & \text{if } \exists \text{ i,j } 1 \leq i \leq k, 1 \leq j \leq T \text{ s.t. } h_i(M_j) = l \\ 0 & \text{otherwise} \end{cases}$$

The final report sent by the CoS to the sink is

$$\{L_E, t, E, i_1, i_2, \cdots, i_T, F\}.$$

Now we need to modify both the en-routing filtering and sink verification procedures to accommodate the use of Bloom filter. When a forwarding node receives the report, it checks whether there are T key indices of distinct partitions and an m-bit Bloom filter F with at most kT "1"s. If it has one of the keys, it re-produces the k hash values and verifies whether the corresponding bits are "1"s. The details of the forwarding protocol for an intermediate sensor node are as follows:

1) Check that T key indices $\{i_j\}$ and an m-bit string F exist in the packet, and there are at most kT "1"s in F; drop the packet otherwise.
2) Check the T key indices $\{i_j, 1 \leq j \leq T\}$ belong to T distinct partitions; drop the packet otherwise.
3) If it has one key $K \in \{K_{i_j}, 1 \leq j \leq T\}$, it computes $M = \overline{MAC}(K, L_E||t||E)$ as in Equation 1. Then it computes each $h_i(M)$ and see if the corresponding bit is "1" in F. The packet is dropped if at least one of them is "0"; it is forwarded to the next hop if all of them are "1".
4) If it does not have any of the keys in $\{K_{i_j}, 1 \leq j \leq T\}$, sends the packet to the next hop.

When the sink receives the report, it checks whether there are T key indices of distinct partitions and a m-bit F with at most kT "1"s in the packet. It then regenerates the Bloom filter and compares with that carried in the packet. Specifically, the sink prepares an m-bit string F', with all bits set to "0". For each key $K \in \{K_{i_j}, 1 \leq j \leq T\}$, it computes the MAC M and marks the corresponding bits $h_i(M), 1 \leq i \leq k$ to "1." The packet is accepted only if F is identical to F'.

Both shorter MACs and Bloom filter can greatly reduce the packet size. As an example, assume that each key index is 10 bits, each MAC is $b = 64$ bits, $T = 5$ key indices and $T = 5$ MACs are required for each report. They take 370 bits (about 46 bytes). Using either shorter MACs, or a Bloom filter of $k = 5$ hash functions, which map 5 MACs to an $m = 64$ bit string, the total required space is reduced to 30% (about 14 bytes). On the other hand, they maintain reasonable levels of security strength and detecting power, as we will compare and analyze in Section IV-B.

IV. Performance Evaluation

In this section we first quantify the effectiveness of en-route filtering in Section IV-A, and compare the security strengths of shorter MACs and Bloom filter, in Section IV-B. Based on the results, we discuss in Section IV-C how to choose appropriate parameters to optimize the detecting power of SEF. We then describe its implementation in Section IV-D and analyze SEF's energy savings through dropping bogus data in Section IV-E. Finally we provide simulation results in Section IV-F.

Since SEF relies on the T carried MACs to detect false reports, an attacker that compromises keys in T or more distinct partitions can successfully fabricate reports. SEF cannot detect or drop such forged reports, which is a limitation of the current design. In the rest of this section, we analyze cases where the attacker has keys in N_c $(0 \leq N_c \leq T-1)$ distinct partitions.

A. En-route filtering effectiveness

The attacker cannot generate correct MACs of other $T - N_c$ distinct categories. To have his data reports being forwarded, the attacker has to forge $T - N_c$ key indices of distinct partitions and $T - N_c$ MACs. We first compute the probability that a forwarding node has one of the $T - N_c$ keys, thus

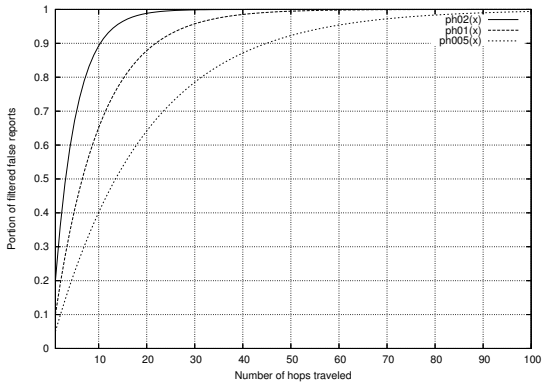

Fig. 5. The portion of dropped false reports as a function of the number of traveled hops. The three curves are for $p_1 = 0.2, 0.1, 0.05$, where the attackers has keys in $1, 3, 4$ distinct partitions, respectively. Each report carries $T = 5$ MACs.

being able to detect an incorrect MAC and drop the report. We analyze the scenarios that use the original MACs here; Section IV-B examines the scenarios using shorter MACs or Bloom filter and shows that the results are almost the same.

If the attacker randomly chooses $T - N_c$ other partitions and randomly chooses a key index in each partition, then the probability that a node happens to have one of the $T - N_c$ keys, denoted by p_1, is

$$p_1 = \frac{T - N_c}{n} \cdot \frac{k}{m} = \frac{k(T - N_c)}{N}, \tag{2}$$

where k is the number of keys each node possesses, m is the number of keys a partition has, and n is the number of key partitions.

The expected fraction of false reports being detected and dropped within h hops is

$$p_h = 1 - (1 - p_1)^h.$$

The average number of hops that a forged report traverses is given as

$$\sum_{i=1}^{\infty} i(1 - p_1)^{i-1} p_1 = \frac{1}{p_1}$$

Figure 5 illustrates how the detected fraction increases as the number of hops h grows. Consider an example of $n = 10$ partitions, $m = 100$ keys per partition, each node stores $k = 50$ keys and each packet carries $T = 5$ MACs. When $N_c = 1, 3, 4$, we have $p_1 = 0.2, 0.1, 0.05$, respectively. Figure 5 shows that 90% false reports are dropped within 10 hops if the attacker has keys in one partition, and 80% are dropped within 15 hops if two partitions. In the worst case, only one MAC is incorrect, 80% false reports are dropped in 32 hops and they travel 20 hops on average.

B. Security Strengths of shorter MACs and Bloom filter

1) False positive at the sink: Now we analyze and compare the false positive probabilities (i.e., a false report is not detected and finally accepted by the sink) of the two overhead-reducing techniques: shorter MACs and Bloom filter. This is important to check whether they maintain sufficient security protection against falsified reports, and identify under what conditions which is better. We assume both techniques use m bits in MACs or a Bloom filter.

Each shorter MAC is of m/T bits, thus the chance that an attacker can successfully guess one MAC, is $2^{-\frac{m}{T}}$. The chance he guesses all the $(T - N_c)$ remaining MACs in addition to the N_c MACs he already knows, thus cheating the sink successfully, is

$$2^{-\frac{m(T-N_c)}{T}}. \tag{3}$$

With Bloom filter, the attacker has two ways to attack. He knows the kN_c hashing results of N_c correct MACs, thus at most kN_c "1"s of an m-bit Bloom filter (Note that more than one result may map to the same bit). Therefore, he needs to guess the remaining $k(T - N_c)$ bit positions of the $(T - N_c)$ forged MACs. Since different hashings may map to the same bit, the $k(T - N_c)$ hash results of forged MACs map to at least one and at most $k(T - N_c)$ distinct bit positions. The total number of bit patterns by $k(T - N_c)$ hash results is

$$B = \sum_{i=1}^{k(T-N_c)} \binom{m}{i} \tag{4}$$

Randomly guessing one of them has $\frac{1}{B}$ chance of success.

A second way of attack is not to mark any additional bit, but to send the Bloom filter of the N_c correct MACs directly. Given the false positive of Bloom filters, it is possible that the "1" bits of the remaining $(T - N_c)$ MACs are already marked. The probability is given by $(1 - e^{-\frac{kN_c}{m}})^{k(T-N_c)}$. Given N_c, m, T, the above probability can be minimized as

$$2^{-\frac{m}{N_c} ln2 (T-N_c)}. \tag{5}$$

Under the first attack, Bloom filter is always better than shorter MACs. This is because the attacker needs to guess in a larger space (B vs. $2^{\frac{m(T-N_c)}{T}}$). As an example, with the same overhead of 64 bits, each report carrying $T = 5$ MACs, in Bloom filter $k = 5$ hash functions are used, whereas each shorter MAC is of 12.8 bit. If the attacker has keys in one partition, the chances of success are 2^{-55} and $2^{-51.2}$; in the worst case when he has keys of $N_c = 4$ distinct partitions, 2^{-23} and $2^{-12.8}$.

Under the second attack, Bloom filter is better when N_c is small, but not as good when N_c is large. By comparing Equations 3 and 5, we find that when the attacker compromises keys in $N_c < Tln2$ partitions, it is far more difficult to succeed in forging the Bloom filter. In the same example, when $N_c = 1$, the probabilities are 2^{-179} and $2^{-51.2}$.

Thus the conclusion is that neither of the two overhead-reducing techniques is always better than the other; they each excels under certain conditions. Note that the above probabilities are *not* the probability of successfully guessing the value of a key, whose strength is decided by its length and independent of the Bloom filter's or shorter MAC's size.

2) False positive at forwarding nodes: An attacker may also try to fool intermediate nodes in order to at least waste energy of sensors along the data delivery paths. With shorter MACs, the chance that an attacker can successfully cheat a

forwarding node without having the correct key, is $2^{-\frac{m}{T}}$. The probability of detecting false reports becomes

$$p_1' = (1 - 2^{-\frac{m}{T}})p_1, \quad (6)$$

where p_1 is the one-hop filtering probability in Equation 2. Since $2^{-\frac{m}{T}} << 1$, the one-hop detecting probability is almost unaffected.

With Bloom filter, the attacker may mark as many bits as possible, trying to cover all the marked bits in a correct Bloom filter. When intermediate nodes find that the bits they calculate are already marked, they will forward the message. Since there are T MACs and each is hashed k times, there are at most kT "1" bits in a correct F. If more than kT bits are "1", an intermediate node can simply drops the report. Thus, the attacker's strategy is: mark the (at most) kN_c bits of the N_c correct MACs, then randomly mark other $k(T - N_c)$ bits as "1". Now we calculate the probability that a forwarding node A with one of the $T - N_c$ keys finds all its bits marked, thus failing to detect such a false report.

Since the hash functions map a MAC to each of the m bits uniformly, the probability that A's k bits all fall in the kT "1"s marked by the compromised node, is $p_c = (\frac{kT}{m})^k$. When $k = \frac{m}{Te}$, this probability can be minimized as $e^{-\frac{m}{Te}}$. For example, when $m = 64$ and $T = 5$, then $p_c \approx 0.01$. Thus the probability of detecting the false report at a forwarding node is

$$p_1' = (1 - e^{-\frac{m}{Te}})p_1, \quad (7)$$

reduced by merely 1%. When $p_1 = 0.2$, we have $p_1' = 0.198$. This implies that 89.0% false reports are filtered out within the initial 10 hops and they travel 5.05 hops on average. Compared with the corresponding results of Section IV-A, the differences are almost negligible.

Therefore, both shorter MACs and Bloom filter retain the en-route filtering power. The difference in their detecting power is almost negligible and they are both effective in defeating an adversary's attack in cheating intermediate forwarding nodes.

C. Parameter Selection

In this section we discuss the impact of parameter choices on SEF effectiveness.

1) Global key pool parameters: The main impact of global key pool structure and key assignment is on en-route filtering. From Equation 2, k/N and T should be large to increase the one-hop detection probability p_1. In practice, k is constrained by the sensor's storage. If each key is 64 bits, storing 50 keys needs 400 bytes. This can take a certain portion of low-end nodes' storage. k/N should not be too big, either. Because each compromised node reveals a portion of the global key pool. With too big a k/N ratio, a few compromised nodes can reveal a significant portion of the key pool.

The choice of T is limited by how many bits the packet can hold. On some low-end nodes, packets cannot be too long, e.g., more than 36 bytes. T should be decided based on the available space excluding the report content, headers, etc. T also affects energy consumption in forwarding. Longer packets consume more energy. We should choose T such that it provides sufficient en-route filtering power while still small enough to conserve energy. We will study its impact on energy in Section IV-E.

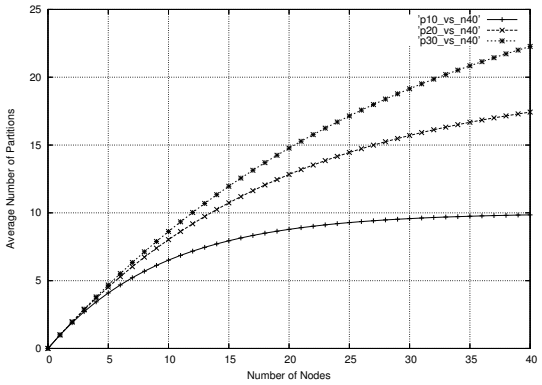

Fig. 6. The average number of partitions $E[n']$ for D nodes. The three curves are for $n = 10, 20, 30$, where n is the total number of partitions

The partition number n affects the en-route filtering probability. A smaller n gives a higher p_1 (Equation 2). On the other hand, n should be larger than T. A larger n makes it more difficult for the attacker to gather keys from all the partitions. The absolute numbers of k, m affect the probability that two nodes have the same set of keys. We should avoid such cases, where compromising one effectively compromises the other. The probability for such cases is determined by the the absolute numbers of k, m, n, given as $1/(n\binom{m}{k})$. Larger k, m lead to smaller probabilities, even though the ratio k/m, thus the filtering probability p_1, remains the same. In practice, a few thousand keys are sufficient to give very small probabilities of two nodes carrying the same set of keys.

2) Deployment density: Another factor we must consider is the node deployment density ρ. Since we require T MACs from distinct categories for each legitimate report, the number of detecting nodes for the same stimulus should be large enough to possess keys from at least T partitions. We can calculate $E[D|n']$, the expected number of nodes needed to collectively possess keys from n' distinct partitions, as follows (details omitted due to space limit):

$$E[D|n'] = n(\frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{n-n'+1}) \approx nln(\frac{n}{n-n'})$$

where n is the total number of partitions. Suppose nodes' sensing radius is r_s, the number of nodes detecting the same stimulus is $N_d = \rho\pi r_s^2$. We should set ρ such at N_d is at least $E[D|T]$ or larger to ensure sufficient number of detecting nodes.

3) Bloom filter parameters: After the key pool parameters are decided, the Bloom filter parameters can be set accordingly. The key index length is $\log_2 N$, thus T key indices and an m-bit Bloom filter take $T \log_2 N + m$ bits for each packet. Within the allowed packet size, m should be large enough to reduce the false positive probabilities, as shown in Equations 4 and 7. The number of hash functions k used in Bloom filter

can be chosen based on the analysis of Equation 7 to minimize the chance of a false data report to escape en-route filtering.

D. Implementation

To help evaluate the performance and complexities of SEF on real sensors, we implemented the communication and cryptography modules of SEF, as well as its APIs, on the MICA2 motes produced by X-Bow [12]. The communication module provides an interface that can be used to efficiently send and receive a packet of any length. The cryptography module facilitates SEF with cryptographic primitives, such as computing the MAC for a given event report, or hashing a MAC for generating Bloom filters.

Each Mica2 sensor is equipped with an 8-bit 4MHz microcontroller running an event-driven operating system, called TinyOS [10], from its internal flash memory. The memory size available at each node is rather limited: 128KB of program memory and 4KB of data memory. These stringent resource constraints clearly require a compact implementation that can fit into the underlying platform. Therefore, we do not intend to provide general-purpose codes in our implementation. Instead, motivated by the application-driven feature of sensor networks, we optimized our codes based on the specific needs of SEF whenever possible.

1) Wireless Communication Module and API: The API exported by the communication module consists of *send()* and *receive()* primitives whose meanings are self-evident. Although TinyOS provides a *GenericComm* interface that can transmit and receive packets, we do not use it and choose to implement our own modules for two reasons. First, the maximum payload size per packet that can be transmitted using this interface is only 29 bytes. Any message that is longer than 29 bytes has to be fragmented. We do not want to constrain the scenarios for our experiments. Second, each packet that goes through this interface is added with an extra 7-byte header, which incurs significant communication overhead if the payload size is small. Some of the fields in the header are not useful for SEF, either. Therefore, we developed a communication module specifically tailored to SEF.

Our communication module directly reads and writes the buffer that is associated with the low-level radio device. It can transmit and receive a packet of any length, though the chance of corruption increases as the packet becomes longer and longer. Moreover, each packet only has a 1-byte header, which contains one field, $Length$. Note that the packet length information is the minimum requirement that should be carried in the packet header, as it is used by the receiver to delimitate the packet. Although this is not appropriate for general-purpose wireless communications, it provides an efficient solution in the context of SEF. Based on the length of a received packet, a node can infer which type of packet it is, and then interpret the packet content accordingly.

2) Cryptography Module and API: We do not intend to implement a complete set of cryptographic primitives in this module. Instead, it provides a simple API, a *MAC(plaintext, key)* call that allows the application to compute the message authentication code for any packet, given a specific symmetric key. Similar to TinySec [13], our implementation of message authentication is based on a single block cipher using RC5 algorithm [14]. RC5 is simple and efficient as it involves addition, XOR, and bit shifting operations only, and has low memory requirement. Hence, it is well suited for providing security in the resource-stringent sensor nodes.

We further optimize our codes by taking advantage of the application semantics. In SEF, the message authentication computation is performed solely on the event content that has a fixed size. Therefore, we need not implement general-purpose MAC computation schemes, such as the CBC-MAC [15], to accommodate input streams with arbitrary lengths. In fact, the event content in our implementation has a size of 8 bytes, which is sufficient to hold the event type, location and detection time and is exactly the length of a block in a 32-bit RC5 block cipher. This way, a single encryption operation suffices to generate the MAC.

To reduce code size, we maximize code reuse by using the same RC5 crypto for hash computations. Implementing conventional hash functions such as MD5 needs separate code and memory to store tables. What we need here is a light-weight function that maps an 8-byte input to a range (e.g., 0-63). We take the first several (e.g., 6) bits of the RC5 output as the hash value. Using the same k keys, each node have the same set of hash functions for Bloom filters.

Module	ROM	RAM
RC5-Crypto	646	128
Radio Stack	4130	114
Combined	4776	242

TABLE I
CODE SIZE BREAKDOWN (BYTES) IN MICA2 PLATFORM

3) Code Breakdown: Table I shows a breakdown of the implementation codes in the MICA2 platform. The cryptography module takes about 0.5% of ROM and 2.8% of RAM, the generic radio communication stack consumes about 3.2% of ROM and 2.8% of RAM. They leave about 124K bytes in ROM and 3.8K bytes in RAM for other components such as applications.

E. Energy Savings

SEF saves energy of sensors along the data dissemination paths through its early detection and dropping of false data reports. On the other hand, SEF requires that each report carry T key indices and MACs (or a Bloom filter), in addition to the normal fields of a report. Such extra fields incur energy consumption in transmission, reception and computation. We use the following model to quantify SEF's energy savings. Let the length of the MACs or Bloom filter, and key index be L_s and L_k, respectively. The length of a normal report without any extra field is denoted as L_r. Then, the length of an SEF report becomes $L'_r = TL_k + L_s + L_r$. We normalize the packet length to L_r and let $\alpha = \frac{L'_r}{L_r} = 1 + \frac{L_s}{L_r} + cT$, where $c = \frac{L_k}{L_r}$. Let the number of hops a report travels be H, and the amount of legitimate data traffic and false injected traffic be 1 and β, respectively. Without SEF, every report (including those

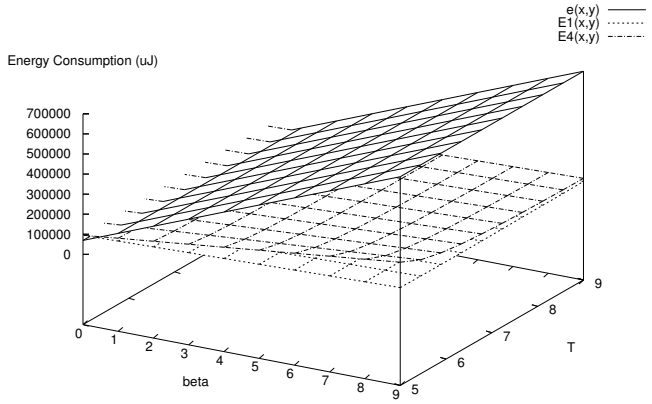

Fig. 7. The energy consumption as a function of the normalized amount of injected traffic β and the number of carried MACs T. e is the energy amount without SEF; $E1, E4$ are the amounts with SEF and the attacker has keys in $1, 4$ distinct partitions, respectively. SEF uses much less energy when the amount of injected traffic exceeds that of legitimate traffic.

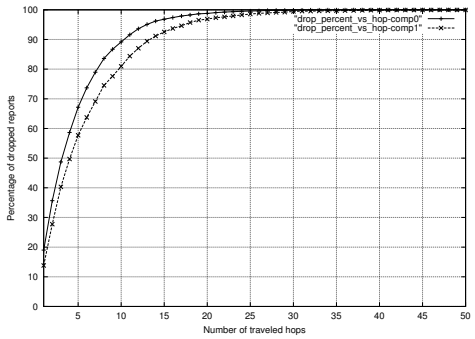

Fig. 8. The percentage of dropped false reports grows as the number of hops increases. The attacker has keys in 0 and 1 partition, respectively.

forged ones) travels all H hops. With SEF, a false report with $T - N_c$ forged MACs has probability $(1-p_1)^{h-1}p_1$ to travel exactly h hops [4], where $p_1 = \frac{k(T-N_c)}{N}$. Therefore, the energy consumed to deliver all the traffic, denoted by e without SEF and E_t with SEF, will be:

$$e = L_r(e_t + e_r)(1+\beta)H \quad (8)$$

$$E_t = L_r(e_t + e_r)(1 + \frac{L_s}{L_r} + cT)(H + \beta\frac{1-(1-p_1)^H}{p_1}) \quad (9)$$

where the energy consumed in transmitting, receiving one byte are e_t, e_r.

Another part of energy of SEF is in computations. We implement RC5 [14] block cipher in Mica2 and use it for both MAC and hash computations. Denote the energy in one hash or MAC computation e_m, the number of hash functions k', the number of detecting nodes for the same stimulus N_d. The computation energy for all the traffic is:[5]

$$E_c = (k'T + N_d)e_m + THe_m + Te_m\beta\frac{1-(1-p_1)^H}{p_1} \quad (10)$$

The total energy consumed for SEF is $E = E_t + E_c$. Measurements [12] show that Mica2 nodes consumes 10mA current when idling or receiving, 13mA transmitting. Based on the battery voltage (3V) and data rate (19.2Kbps), we can calculate that it takes $e_t = 16.25, e_r = 12.5$ μJ to transmit/receive a byte. Each RC5 computation takes about 0.5 ms [16] and consumes about $e_m = 15$ μJ. Since the implementation uses the same RC5 crypto for both MAC and hash computations, it takes 0.5 ms to finish one MAC computation (2.5ms to finish the 5 hash computations for the Bloom filter).

[4]The actual per hop detecting probability should be $p_1' = p_1(1-p_c)$. Since p_c is very small (Section IV-B), we ignore it in the computation.

[5]This is for SEF with Bloom filter. For shorter MACs there is no hash computation, thus the energy is even less. We omit it due to space limit.

Figure 7 plots how e and E change as functions of different T and β, when $H = 100$, the overhead of Bloom filter ($k' = 5$ hash functions are used) or shorter MACs is $L_s = 64$ bits, key index is $L_k = 10$ bits, original packet size is $L_r = 24$ bytes, the global key pool has 10 partitions and a node has 50% of the keys in a partition, $N_d = 10$ nodes detect the same stimulus. SEF energy is plotted for two cases, the attacker has keys in $N_c = 1, 4$ distinct partitions.

We find that e grows much faster than E, and SEF saves energy in most cases. For example, when $\beta = 9$, if the packet carries $T = 5$ MACs and the attacker has keys in one partition, with SEF more than 80% energy can be saved compared to the case without SEF. Even with the worst case of $N_c = 4$, where the one-hop filtering probability p_1 is merely 0.05, still about 65% of the total energy can be saved by dropping false reports.

In reality, an attacker may inject false data reports that are orders of magnitude more than legitimate traffic, to inflict severe damage to the network. SEF saves significant amount of energy in these scenarios. When the number of false reports is low, SEF may not save as much energy. But it still enables the sink to detect bogus events and reduce false alarms.

We also find that E_t dominates SEF's total energy consumption, which is consistent with the observation that it saves energy to trade off computation for less communication. An additional discovery is that T should not be too small to reduce energy consumption. Otherwise the one-hop filtering probability is decreased and injected reports will travel more hops and consume more energy.

F. Simulation Results

We use simulations to further verify our analysis. Due to space constraint, we only present results for en-route filtering and energy consumption in cases of $N_c = 0, 1$. We use a field size of $200\times20m^2$ where 340 nodes are uniformly distributed. One stationary sink and one stationary source sit in opposite ends of the field, with about 100 hops in between. The power consumptions of transmission and reception are 60mW and 12mW, respectively. The transmission time for a packet is 10 ms. The source generates a report every two seconds. We use a global key pool of 1000 keys, divided into 10 partitions, with 100 keys in each partition. Each node has 50 keys. The results are averaged over 10 simulated topologies.

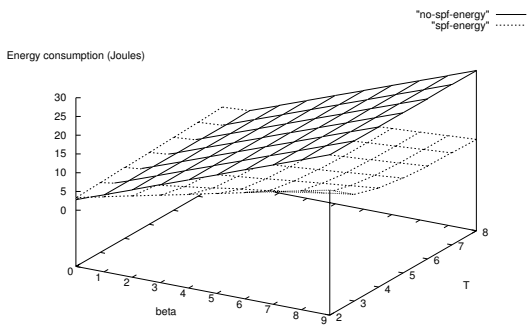

Fig. 9. The energy consumption as a function of injected traffic ratio β and the number of MACs T each report carries. The attacker has keys in one partition. SEF saves energy when the amount of injected traffic exceeds that of legitimate traffic

a) En-route filtering: Figure 8 shows the percentage of dropped false reports as a function of the number of traveled hops, for 0 and 1 compromised key partition, respectively. The source generates 1000 bogus reports in each run. When the attacker mimics wireless transmission to inject traffic, about 90% false reports are dropped within 10 hops. With one compromised node, about 80% are dropped within 10 hops. As the reports travel, more and more are detected and dropped: less than 5% reports can go beyond 20 hops and none reaches the sink - all of them are detected and dropped before they finish half of the 100 hops. These results are consistent with the theoretical analysis. They show that SEF turns the network scale into an asset to achieve greater detection power with a larger node population.

b) Energy Tradeoff: We use similar parameters as those in Section IV-E and the attacker has keys in one partition. The source generates 100 reports. The number of forged reports is 100β, where β is the injected traffic ratio. Figure 9 confirms our previous analysis (Section IV-E): when the injected traffic is more than the legitimate traffic, SEF saves energy.

V. Discussions and Future Work

A. Other network factors

SEF turns the scale into an asset by accumulating detecting power over data delivery paths. It works the best in large scale networks where reports need to travel many hops to reach the sink. The longer the data delivery path, the more powerful the en-route filtering. SEF may not filter reports injected from locations close to the sink effectively. In such cases the energy savings might not be significant. However, the sink's detecting power is not affected. Any forged reports with incorrect MACs are still detected and do not trigger false alarms.

SEF design is not tied to any particular data forwarding protocol. The probabilistic key assignment allows any node to verify MACs regardless of its location or functionality. As long as sensors can store some keys and perform the MAC computations, injected false reports can be detected and filtered en-route. It works with existing data forwarding protocols such as Directed Diffusion [17], GRAB [8] and TTDD [18].

SEF requires that data reports be generated collaboratively by *multiple* detecting sensors of the same stimulus. If the node density is low, detecting nodes may not generate sufficient numbers of MACs. We expect the guidelines in Section IV-C to help setting appropriate deployment densities to avoid such situations.

The topology of a sensor network may change dynamically due to energy conserving protocols such as [19], [20], or unexpected node deaths in a harsh environment. SEF does not need any extra mechanism to deal with topology changes, because the state used for detection, key indices and MACs, is carried in packets, not by sensors.

B. Future work

When nodes are densely deployed, the detecting nodes of a stimulus can collectively hold keys of all the n partitions. In this scenario if the attacker has keys belonging to T or more, but less than n partitions, we may still detect their false reports over time. Since CoS can choose to load consecutive data reports with MACs from different sets of T categories, a few successive reports will collectively carry MACs of all n categories. The sink can then reject those stimuli whose reports have MACs of less than a threshold T_{th} ($T < T_{th} \leq n$) of distinct categories. Further exploring the *temporal redundancy* of sensor network data reports in addition to the *spatial redundancy*, SEF can detect false data reports injected by attackers holding keys of up to $n - 1$ distinct partitions.

A compromised detecting node may launch false negative attacks against the collaborative report generation process. Thus a legitimate event may not be reported properly. This is a different problem from false positives that SEF addresses. Besides, being elected as the CoS is non-trivial. Unless the compromised node broadcasts values that are consistent with other detecting sensor nodes' observation, effectively reporting the truth, they will not follow and endorse the compromised node's data report. Another legitimate detecting node will announce its values and be elected.

One way to improve the resilience of the CoS election against attacks and detect incorrect MACs is to add pairwise keys shared between each pair of neighbors and require each message be authenticated. Thus one compromised node can send at most one incorrect MAC to the CoS, but cannot impersonate other nodes. The probabilistic key assignment of SEF already allows certain degrees of key sharing between neighbors to establish pairwise keys during network bootstrapping.

Currently SEF does not address identifying compromised nodes or replacing compromised keys, which may be needed for the continuous operation of a network. For identification, neighbor nodes may overhear the channel to detect unusual activities of compromised nodes such as high traffic volume and notify the sink. After the nodes are identified, the sink should flood instructions to revoke compromised keys and propagate new ones. The sink's instructions can be authenticated by hash chains as proposed in [1]. As more nodes are compromised or deplete energy, new nodes should be deployed. To work with existing nodes, they can carry some keys from the same global

key pool. However, those keys that are already compromised should be avoided.

Finally, SEF is not designed to address other attacks a compromised node may launch, such as dropping legitimate reports passing through it, recording and replaying legitimate reports, or injecting false *control* packets to disrupt other protocols. Certain techniques can be applied. The authors in [21] point out multipath forwarding can alleviate dropping of legitimate reports. Each sensor can use a cache [8], [17] to store the signatures of recently forwarded reports, thus replayed packets are not forwarded again. Solving these various problems requires different solutions.

We are currently exploring a location dependent key management scheme to confine the impact of a compromised sensor node to its locality. In the new scheme, keys are only valid in endorsing data reports regarding events in certain locations. Therefore, an attacker has to compromise sensor nodes *locally* and accumulate T or more partitions in order to have its inject false data reports forwarded by the sensor network.

VI. Related Work

Sensor network security has been studied in recent years in a number of proposals. Karlof et al. [21] analyzes attacks against sensor network routing protocols and points out possible ways of defense. Wood et al. [22] studies DoS attacks against different layers of sensor protocol stack and concludes that security should be considered at the design phase. Sasha et al. [23] proposes to trade-off overhead and security strength based on the importance of data. Lin et al. [24] studies how to reduce energy consumption in cryptographic algorithms using dynamic voltage scaling. Carman et al. [7] compares the energy consumptions of different public key algorithms on various sensor hardware. SEF addresses a different problem of detecting and en-route filtering injected false data.

SPINS [1] implements symmetric key cryptographic algorithms with delayed key disclosure on Motes to establish secure communication channels between a base station and sensors within its range. Basagni et al. [6] uses a single "mission key" for the entire sensor network assuming that tamper-resistant hardware is available so that no secret can be compromised. They do not address the false data injection problem in the presence of compromised sensor nodes.

SEF key assignment bears similarities with [2], [3], which use probabilistic key sharing to establish trust between neighboring nodes. Chan et al. [3] further trades off the unlikelihood of large scale attacks for higher strength against smaller ones. But SEF solves a different problem, and it assigns keys differently: each node has keys from only one partition of the global pool. This is to ensure each node can only generate part of the proof for the truthfulness of a report. Only through the joint effort of multiple nodes can the complete proof be generated. Eschenauer et al. [2] does not impose such a constraint and nodes can choose keys from the whole key pool. Finally, [2] requires any two nodes have very high probabilities of sharing keys to build a connected network; the probability in SEF can be much lower since we exploit the network scale to make en-route filtering effective.

In principle, the joint generation of MACs by multiple nodes is similar to [4] where several nodes collectively issue a certificate for a new node in mobile ad hoc networks. But [4] uses public key algorithms, which are infeasible on small sensors of constrained computing, energy and memory resources. Canetti et al. [25] proposes multiple MACs to ensure source authentication in multicast so that a group of less than a threshold number of colluding receivers do not have all the keys needed to cheat other receivers. SEF has many data sources but one sink and only the sink has all the keys. The purpose is to prevent compromised nodes from cheating the sink. [25] assumes one source but many receivers and the purpose is to prevent cheating each individual receiver. Also, packet size is not a big concern in the Internet but it is a serious issue for low-end sensors.

There is also a rich literature on secure routing in mobile ad-hoc networks such as [26], [27]. Yang et al. [28] proposes self-organized algorithms and protocols to secure homogeneous ad-hoc wireless networks, and Kong et al. [29] for heterogeneous mobile ad-hoc networks. SEF works for large-scale sensor networks, whose communication is usually from many to one and the resources are severely constrained.

SEF design is also related to intrusion detection [30] and Internet packet filtering against DoS attacks through forged source IP addresses [5]. However, these designs either rely on the network infrastructure that does not exist in a self-organized wireless sensor network, or involve complex and sophisticated mechanisms that are beyond the capabilities of low-end sensors. SEF only requires that sensor nodes store tens of keys and perform efficient keyed MAC computations.

VII. Conclusion

Large scale sensor network may be deployed in a potentially adverse or even hostile environment. Due to the unattended operations of the network and the relatively small sizes of the sensors, sensor nodes may have a high risk of being captured and compromised. Instead of relying on, and complementing the efforts of, tampering prevention, in this paper we focus on detecting false sensing reports that can be injected by compromised nodes.

We have developed a Statistical En-route Filtering mechanism (SEF) for false report detection. Authenticating event reports requires that nodes share certain security information, however attackers can obtain such information by compromise just a single node. To overcome this dilemma SEF design divides a global key pool into multiple partitions and carefully assigns a certain number of keys from one partition to individual node. Given any single node knows only a limited amount of the system secret, compromising of one or a small number of nodes cannot disable the overall network from detecting bogus reports. SEF design harnesses the advantage of large scale by requiring endorsement of an event report from multiple detecting nodes and by detecting false reports through collaborative filtering of all forwarding nodes along the path. Our analysis and simulations show that SEF can drop up to 70% of bogus reports injected by a compromised node within 5 hops, and up to 90% within 10 hops along the forwarding

paths. When the amount of injected traffic is high, it saves more than 80% of energy by dropping false data en-route.

Although several recent research efforts have addressed sensor network security issues such as node authentication, data secrecy and integrity, they provide no protection once any *single* node is compromised. SEF represents the first step towards building resilient sensor networks that can withstand *compromised* nodes. SEF achieves this goal by balancing the tradeoff between the amount of security information assigned to individual nodes and the false detection power of the nodes. The more security information each forwarding node has, the more effective the en-route filtering will be, but also the more secret the attacker can obtain from a compromised node. Our plan for the next step is to conduct systematic evaluation of the tradeoffs between these two conflict goals, and to gain further insight in how to build a sensor network that can be at once resilient against multiple compromised nodes as well as effective in detecting false data reports through collaborative filtering.

REFERENCES

[1] V. Wen, A. Perrig, and R. Szewczyk, "SPINS: Security Suite for Sensor Networks," in *ACM MOIBCOM*, 2001.

[2] L. Eschenauer and V. D. Gligor, "A Key-Management Scheme for Distributed Sensor Networks," in *ACM CCS*, 2002.

[3] H. Chan, A. Perrig, and D. Song, "Random Key Predistribution Schemes for Sensor Networks," in *IEEE Symposium on Security and Privacy*, 2003.

[4] H. Luo, J. Kong, P. Zerfos, S. Lu, and L. Zhang, "URSA: Ubiquitous and Robust Access Control for Mobile Ad-Hoc Networks," *to appear in IEEE/ACM Transactions on Networking*, October 2004.

[5] K. Park and H. Lee, "On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets," in *ACM SIGCOMM*, 2001.

[6] S. Basagni, K. Herrin, E. Rosti, and D. Bruschi, "Secure Pebblenets," in *ACM MOBIHOC*, 2001.

[7] D. W. Carman, P. S. Kruus, and B. J. Matt, "Constraints and Approaches for Distributed Sensor Network Security," NAI Labs, Tech. Rep. 00-010, September 2000.

[8] F. Ye, G. Zhong, S. Lu, and L. Zhang, "GRAdient Broadcast: A Robust Data Delivery Protocol for Large Scale Sens or Networks," *ACM Wireless Networks (WINET)*, vol. 11, no. 2, March 2005.

[9] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Crypto*, 1996.

[10] "TinyOS Operation System," http://millennium.berkeley.edu.

[11] B. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, 1970.

[12] "Xbow sensor networks," http://www.xbow.com/.

[13] C. Karlof, N. Sastry, U. Shankar, and D. Wagner, "TinySec: TinyOS Link Layer Security Proposal - version 1.0," 2002.

[14] R. Rivest, "The RC5 Encryption Algorithm," in *Workshop on Fast Software Encryption*, 1995.

[15] U. N. I. of Standards and Technology, "Data Encryption Standard (DES)," Draft Federal Information Processing Standards Publication 46-3, 1999.

[16] C. Karlof, N. Sastry, and D. Wagner, "TinySec: Security for TinyOS," www.cs.berkeley.edu/~nks/tinysec/TinySec.ppt, 2002.

[17] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," in *ACM MOBICOM*, 2000.

[18] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang, "A Two-tier Data Dissemination Model for Large-scale Wireless Sensor Networks," in *ACM MOIBCOM*, 2002.

[19] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. B. Srivastava, "Optimizing Sensor Networks In The Energy-Latency-Density Design Space," *Ieee Transactions On Mobile Computing*, vol. 1, no. 1, 2002.

[20] F. Ye, G. Zhong, S. Lu, and L. Zhang, "PEAS: A Robust Energy Conserving Protocol for Long-lived Sensor Networks," in *ICDCS*, 2003.

[21] C. Karlof and D. Wagner, "Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures," in *IEEE SPNA*, 2002.

[22] A. Wood and J. Stankovic, "Denial of Service in Sensor Networks," *IEEE Computer*, October 2002.

[23] S. Slijepsevic, M. Potkonjak, V. Tsiatsis, S. Zimbeck, and M. Srivastava, "On Communication Security in Wireless Ad-Hoc Sensor Networks," in *11th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2002.

[24] L. Yuan and G. Qu, "Design Space Exploration for Energy-Efficient Secure Sensor Networks," in *IEEE International Conference on Application-Specific Systems, Architectures and Processors*, 2002.

[25] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas, "Multicast Security: A Taxonomy and Some Efficient Constructions," in *INFOCOM*, 1999.

[26] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Ariadne: A Secure On-demand Routing Protocol for Ad Hoc Networks," in *ACM MOBICOM*, 2002.

[27] B. Awerbuch, D. Holmer, C. Nita-Rotaru, and H. Rubens, "An On-Demand Routing Protocol Resilient to Byzantine failures," in *ACM Workshop on Wireless Security (WiSe)*, 2002.

[28] H. Yang, X. Meng, and S. Lu, "Self-organized Network Layer Security in Mobile Ad Hoc Networks," in *WiSe*, 2002.

[29] J. Kong, H. Luo, K. Xu, D. L. Gu, M. Gerla, and S. Lu, "Adaptive Security for Multi-layer Ad Hoc Networks," *Wireless Communications and Mobile Computing, Special Issue on Mobile Ad Hoc Networking*, vol. 2, pp. 533–547, 2002.

[30] W. R. Cheswick and S. M. Bellovin, *Firewalls and Internet Security*. Addison-Wesley, 1994.

Fan Ye Fan Ye received his B.E. in Automatic Control in 1996 and M.S. in Computer Science in 1999, both from Tsinghua University, Beijing, China. He received his Ph.D. in Computer Science in 2004 from UCLA. He is currently with IBM Research. His research interests are in wireless networks, sensor networks and security.

Haiyun Luo Haiyun Luo received his B.S. degree from University of Science and Technology of China, and his M.S. and Ph.D. degrees in Computer Science from University of California at Los Angeles. He is currently an assistant professor in Department of Computer Science, University of Illinois at Urbana-Champaign. His research interests include wireless and mobile networking and computing, security and large-scale distributed systems.

Songwu Lu Songwu Lu received both his M.S. and Ph.D. from University of Illinois at Urbana-Champaign. He is currently an assistant professor at UCLA Computer Science. He received NSF CAREER award in 2001. His research interests include wireless networking, mobile computing, wireless security, and computer networks.

Lixia Zhang Lixia Zhang received her Ph.D in computer science from the Massachusetts Institute of Technology. She was a member of the research staff at the Xerox Palo Alto Research Center before joining the faculty of UCLA's Computer Science Department in 1995. In the past she has served on the Internet Architecture Board, Co-Chair of IEEE Communication Society Internet Technical Committee, the editorial board for the IEEE/ACM Transactions on Networking, and technical program committees for many networking-related conferences including SIGCOMM and INFOCOM. Zhang is currently serving as the vice chair of ACM SIGCOMM.