Catching Packet Droppers and Modifiers in Wireless Sensor Networks

Chuang Wang*, Taiming Feng*, Jinsook Kim*, Guiling Wang**, and Wensheng Zhang*
*Department of Computer Science, Iowa State University
**Department of Computer Science, New Jersey Institute of Technology
Emails: {cwang,taiming,dvorakjs,wzhang}@cs.iastate.edu, gwang@njit.edu

***Abstract*—Packet dropping and modification are common attacks that can be launched by an adversary to disrupt communication in wireless multi-hop sensor networks. Many schemes have been proposed to mitigate the attacks but none can effectively and efficiently identify the intruders. To address the problem, we propose a simple yet effective scheme, which can identify misbehaving forwarders that drop or modify packets. Extensive analysis and simulations using ns2 simulator have been conducted and verified the effectiveness and efficiency of the scheme.**

I. Introduction

In a wireless sensor network, sensor nodes monitor the environment, detect events of interest, produce data and collaborate in forwarding the data towards a sink, which could be a gateway, base station, storage node, or querying user. A sensor network is often deployed in an unattended and hostile environment to perform the monitoring and data collection tasks. When it is deployed in such an environment, it lacks physical protection and is subject to node compromise. After compromising one or multiple sensor nodes, an adversary may launch various attacks [1] to disrupt the in-network communication. Among these attacks, two common ones are *dropping packets* and *modifying packets*, i.e., compromised nodes drop or modify the packets that they are supposed to forward.

To deal with packet droppers, a widely adopted countermeasure is multi-path forwarding [2], [3], in which each packet is forwarded along multiple redundant paths and hence packet dropping in some but not all of these paths can be tolerated. This scheme introduces high extra communication overhead. Another category of countermeasures is to monitor the behavior of forwarding nodes [4]–[6]. However, these schemes are subject to high energy cost incurred by the promiscuous operating mode of wireless interface. To deal with packet modifiers, most of existing countermeasures [7]–[10] are to filter modified messages within a certain number of hops. However, without identifying packet droppers and modifiers, these countermeasures can not fully solve the packet modification problems because the compromised nodes can continue attacking the network without being caught. To identify packet modifiers, Ye et al. [11] recently proposed a probabilistic nested marking (PNM) scheme to identify packet modifiers with a certain probability. However, the PNM scheme cannot be used together with the false packet filtering schemes [7]–[10], because the filtering schemes will drop the modified packets which should be used by the PNM scheme as evidences to infer packet modifiers. This degrades the efficiency of deploying the PNM scheme.

In this paper, we propose a simple yet effective scheme to catch both packet droppers and modifiers. According to the scheme, a dynamic routing tree rooted at the sink is first established. When sensor data is transmitted along the tree structure towards the sink, each packet sender or forwarder adds a small number of extra bits, which is called packet marks, to the packet. The format of the small packet marks is deliberately designed such that the sink can obtain very useful information from the marks. Specifically, based on the packet marks, the sink can figure out the dropping rate associated with every sensor node, and then run our proposed *node categorization algorithm* to identify nodes that are droppers/modifiers for sure or are suspicious droppers/modifiers. As the tree structure dynamically changes every certain time interval, behaviors of sensor nodes can be observed in a large variety of scenarios. As the information of node behaviors has been accumulated, the sink periodically run our proposed *heuristic ranking algorithms* to identify most likely bad nodes from suspiciously bad nodes. This way, most of the bad nodes can be gradually identified with small false positive.

Compared with existing schemes, our scheme has the following unique characteristics: (1) being effective in identifying both packet droppers and modifiers, (2) low overhead in terms of both communication and energy consumption, and (3) being compatible with existing false packet filtering schemes [7]–[10]; that is, it can be deployed together with the false packet filtering schemes, and therefore can not only identify intruders but also filter modified packets immediately after the modification is detected. Extensive simulation on ns2 simulator have been conducted to verify the effectiveness and efficiency of the proposed scheme in various scenarios.

The rest of the paper is organized as follows: Section II defines the system model. Section III describes the proposed scheme and section IV reports the evaluation results. Section V concludes the paper.

II. System Model

A. Network Assumptions

We consider a typical deployment of sensor network, as shown in Fig. 1, where a large number of sensor nodes are deployed in a two dimensional area. Each sensor node

978-1-4244-2908-0/09/$25.00 ©2009 IEEE

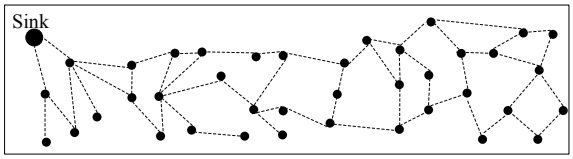


Fig. 1. Network System Model

generates sensing data periodically and all these nodes collaborate to forward packets that contain the data hop by hop towards a sink. The sink is located at some place within the network. We assume that all sensor nodes and the sink are time synchronized [12], which is required by many applications. The sink is aware of the network topology, which can be achieved by requiring nodes to report their neighboring nodes soon after deployment.

B. Security Assumptions and Attack Model

We assume the network sink is trustworthy and free of compromise, but regular sensor nodes can be compromised. Compromised nodes may or may not collude with each other. A compromised node can launch the following two attacks:

- Packet dropping: a compromised node drops all or some of the packets that it is supposed to forward. It may also drop the data generated by itself for some malicious purpose such as accusing innocent nodes.
- Packet modification: a compromised node modifies all or some of the packets that it is supposed to forward. It may also modify the data it generates to protect itself from being identified or to accuse other nodes.

III. The Proposed Scheme

In our scheme for identifying packet droppers and modifiers, a system initialization phase is followed by several equal-duration rounds of intruder identification phases.

- In the initialization phase, sensor nodes form a dynamic routing tree rooted at the sink. The structure of the tree changes dynamically from round to round.
- In each round, data traffic is transmitted through the routing tree to the sink, and each packet sender/forwarder adds a small number of extra bits to the packet and also encrypts the packet. When one round finishes, based on the extra bits carried in the received packets, the sink runs the node categorization algorithm to identify nodes that must be droppers or modifiers and nodes that are suspiciously bad.
- The routing tree is reshaped every round. As a certain number of rounds have passed, the sink will have collected information about node behaviors in different routing topologies. The information includes which nodes are bad for sure, which nodes are suspiciously bad, and the nodes' topological relationship. To further identify bad nodes from the potentially large number of suspiciously bad nodes, the sink runs heuristic ranking algorithms.

In the following sub-sections, we first present the algorithm for tree establishment and packet transmission, which is followed by our proposed categorization algorithm, tree structure reshaping algorithm, and heuristic ranking algorithms. To ease the presentation, we first concentrate on packet droppers and assume no node collusion. After that, we present how to extend the presented scheme to handle node collusion and detect packet modifiers, respectively.

A. Tree Establishment and Packet Transmission

All sensor nodes form a tree rooted at the sink. The sink knows the tree topology and shares a unique key with each node on the tree. When a node wants to send out a packet, it attaches to the packet a sequence number, encrypts the packet with the key shared with the sink, and then forwards the packet to its parent. When an innocent intermediate node receives a packet, it attaches a few bits to the packet to mark the forwarding path of the packet, encrypts the packet, and then forwards the packet to its parent. On the contrary, a misbehaving intermediate node may drop a packet it receives. On receiving a packet, the sink decrypts it, and thus finds out the original sender and the packet sequence number. The sink keeps tracking the sequence numbers of received packets for every node, and for every certain time interval, which we call a *round*, it calculates the packet dropping rate for every node. Based on the dropping rate and the knowledge of tree topology, the sink identifies packet droppers based on rules we derive. In detail, the scheme includes the following components, which are elaborated in the following.

1) System Initialization: The purpose of system initialization is to set up secret pair-wise keys between the sink and every regular sensor node, and to establish a tree to facilitate packet forwarding from every sensor node to the sink.

Preloading Keys and Other System Parameters

Each sensor node u is preloaded the following information:

- K_u: a secret key exclusively shared between the node and the sink.
- N_p: the maximum number of candidate parent nodes that each sensor node records during the tree establishment procedure.
- N_s: the maximum packet sequence number. For each sensor node, its first packet has sequence number 0, the N_s^{th} packet is numbered $N_s - 1$, the $(N_s + 1)^{th}$ packet is numbered 0, and so on and so forth.

Tree Establishment

After deployment, the sink broadcasts to its one-hop neighbors a 2-tuple $\langle 0, 0\rangle$. In the 2-tuple, the first field is the ID of the sender (We assume the ID of sink is 0.) and the second field is its distance in hop from the sender to the sink. Each of the remaining nodes, assuming its ID is u, acts as follows:

(i) On receiving the first 2-tuple $\langle v, d_v\rangle$, node u sets its own distance to the sink as $d_u = d_v + 1$.

(ii) Node u records each node w (including node v) as its parent if it has received $\langle w, d_w\rangle$ where $d_w = d_v$. If the number of recorded parents is greater than N_p, only N_p parents are kept, and others are discarded. The actual number of parents it has recorded is denoted by $n_{p,u}$.

(iii) After a certain time interval[1], node u broadcasts 2-tuple $\langle u, d_u \rangle$ to its one-hop neighbors. Then, among the recorded parents, node u randomly picks one (whose ID is denoted as P_u) as its actual parent, and a random number (which is denoted as R_u) between 0 and $N_p - 1$. Finally, node u sends P_u, R_u and all $n_{p,u}$ recorded parents to the sink.

After the above procedure completes, a tree rooted at the sink is established. The sink also knows the topology of the trees from the reports it has received from every node. Note that the sink not only knows the current tree structure, but also the parents recorded by each node.

2) Packet Sending and Forwarding: Each node maintains a counter C_p which keeps track of the number of packets that it has sent so far. When a sensor node u has a data item D to report, it composes and sends the following packet to its parent node P_u:

$$\langle P_u, \{R_u, u, C_p \text{ MOD } N_s, D, pad_{u,0}\}_{K_u}, pad_{u,1} \rangle,$$

where C_p MOD N_s is the sequence number of the packet. R_u is a random number between 0 and $N_p - 1$ picked by node u during tree establishment. $\{X\}_Y$ represents the result of encrypting X using key Y.

Padding $pad_{u,0}$ and $pad_{u,1}$ are added to make all packets equal length, such that forwarding nodes cannot tell packet sources based on packet length. Meanwhile, the sink can still decrypt the packet to find out the actual content. To satisfy these two objectives simultaneously, the padding is constructed as follows:

- For a packet sent by a node which is h hops away from the sink, the length of $pad_{u,1}$ is $\log(N_p) * (h-1)$ bits. As to be described later, when a packet is forwarded for one hop, $\log(N_p)$ bits information will be added and meanwhile, $\log(N_p)$ bits will be chopped.
- Let the maximum size of a packet be L_p bits, a node ID be L_{id} bits and data D be L_D bits. $pad_{u,0}$ should be $L_p - L_{id} * 2 - \log(N_p) * h - \log(N_s) - L_D$ bits, where $L_{id} * 2$ bits are for P_u and u fields in the packet, field R_u is $\log(N_p)$ bits long, field $pad_{u,1}$ is $\log(N_p) * (h-1)$ bits long, and C_p MOD N_s is $\log(N_s)$. Setting $pad_{u,0}$ to this value ensures that all packets have the same size L_p.

When a sensor node v receives packet $\langle v, m \rangle$, it composes and forwards the following packet to its parent node P_v:

$$\langle P_v, \{R_v, m'\}_{K_v} \rangle,$$

where m' is obtained by trimming the rightmost $\log(N_p)$ bits off m. Meanwhile, R_v, which has $\log N_p$ bits, is added to the front of m'. Hence, the size of the packet keeps unchanged.

3) Packet Receipt at the Sink: We use node 0 to denote the sink. When the sink receives a packet $\langle 0, m \rangle$, it conducts the following steps:

[1]The length of the interval is a predefined system parameter that is large enough for each node to receive an enough number of broadcasts from the nodes closer to the sink.

(i) Initialization: We introduce two temporary variables u and m'. Let $u = 0$ and $m' = m$.
(ii) The sink attempts to find out a child of node u, denoted as v, such that $dec(K_v, m)$ results in a string starting with R_v, where $dec(K_v, m)$ means the result of decrypting m with key K_v.
(iii) If the attempt fails, the packet is identified as being modified and thus should be dropped.
(iv) If the attempt succeeds, it indicates that the packet was forwarded from node v to node u. Now, there are two cases:
  – If $dec(K_v, m)$ starts with $\langle R_v, v \rangle$, it indicates that node v is the original sender of the packet. The sequence number of the packet is recorded for further calculation and the receipt procedure completes.
  – Otherwise, it indicates that node v is an intermediate forwarder of the packet. Then, u is updated to be v, m' is updated to be the string obtained by trimming R_v from the leftmost. Then, steps (ii)-(iv) are repeated.

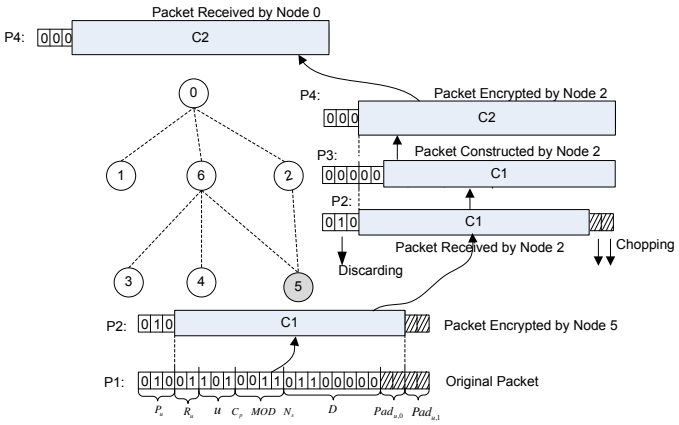


Fig. 2. Example of Packet Sending, Forwarding

4) An Example: Fig 2 shows an example sensor network with 7 nodes, node $0-6$. Node ID is represented by 3 bits. Suppose the maximum packet sequence number N_s is 16 and 4 bits are used to represent the counter C_p. N_p, the maximum number of parents that each sensor node should record during the tree establishment, is 4. We assume that the length of sensory data L_D is 8 bits. In this figure, we illustrate the procedure when node 5, which is 2 hops away from the sink, generates sensory data 96, how the data is sent to the sink node 0. Assume currently, data from node 5 follows path 5->2->0 and $C_p = 3$. Node 5 constructs packet

$$< P_u, \{R_u, u, C_p \ MOD \ N_s, D, pad_{u,0}\}_{K_u}, pad_{u,1} >,$$

The plain-text of the packet is shown in the figure as P1. Specifically, $P_u = 2(010)$, $R_u = 1(01)$, $u = 5(101)$, $C_p = 3(0011)$, and $D = 96(01100000)$. The length of the paddings are calculated as follows. Assume that the maximum packet size L_p is 24 bits. The length of $pad_{5,1}$ should be $\log N_p * (h-1)$ bits, that is 2 bits. The length of $pad_{5,0}$ should be

$L_p - L_{id} * 2 - \log N_p * h - \log N_s - L_D$, that is, $24 - 3 * 2 - 2 * 2 - 4 - 8 = 2$ bits. Based on P1, node 5 uses its secret key K_5 to encrypt part of $P1$, $\{R_5, 5, C_p \ MOD \ N_s, D, pad_{5,0}\}$. The cipher-text is represented by $C1$ and the encrypted packet $P2$ is constructed accordingly. P_2 is sent to node 2.

When node 2 receives packet $P2$, it first chops the rightmost $\log N_p$ bits, which is 2 bits of the paddings. Next, node 2 constructs packet $P3$ by adding its parent ID and the random number R_2 to the front of cipher-text $C1$. Note that the packet length is kept the same since the right most 2 bits is chopped, and 2 bits random number R_2 is added. Next, node 2 uses its secret key K_2 to encrypt information $\{R_2, C1\}$ in packet $P3$ and generates packet $P4$. $P4$ is then sent to the sink.

After the sink receives the packet $P4$ from its children, the sink tries to figure out the sender. The sink tries to decrypt the cipher-text $C2$ by using its children's secret keys one by one. The sink finds that the packet is from node 2 after $C2$ is decrypted by using K_2. The sink also recovers the decrypted $C2$ which does not starts with $\{R_2, 2\}$. (Note that, the sink and each sensor node are synchronized and they follow an implicit tree reshaping algorithm. The random number R_2 is also known by the sink.) The sink concludes that node 2 is an intermediate node. It continues this process and finds out the source of the data is node 5.

B. Node Categorization Algorithm

In every round, for each sensor node u, the sink keeps track of the number of packets sent from u, the sequence numbers of these packets and the number of flips in the sequence numbers of these packets, (i.e., the sequence number changes from a large number such as $N_s - 1$ to a small number such as 0). In the end of each round, the sink calculates the dropping rate for each node u. Suppose $n_{u,max}$ is the most recently seen sequence number, $n_{u,flip}$ is the number of sequence number flips and $n_{u,rcv}$ is the number of received packets. The dropping ratio in this round is calculated as follows:

$$\frac{n_{u,flip} * N_s + n_{u,max} + 1 - n_{u,rcv}}{n_{u,flip} * N_s + n_{u,max} + 1}.$$

Based on the dropping rate of every sensor node and the tree topology, the sink identifies the nodes that are droppers for sure and that are possibly droppers. For this purpose, a threshold θ is first introduced. We assume that if a node's packets are not intentionally dropped by forwarding nodes, the dropping rate of this node should be lower than θ. Note that θ should be greater than 0, taking into account droppings caused by incidental reasons such as collisions. The first step of the identification is to mark each node with "+" if its dropping ratio is lower than θ, or with "-" otherwise. After all nodes have been marked with "+" or "-", we can identify the following patterns for each node, which are also illustrated by Fig. 3:

- $+\{+\}^+$: The node and its one or more continuous immediate upstream nodes are marked as "+".
- $+\{-\}^+$: The node is marked as "+", but its one or more continuous immediate upstream nodes are marked as "-".
- $-\{+\}^+$: The node is marked as "-", but its one or more continuous immediate upstream nodes are marked as "+".
- $-\{-\}^+$: The node and its one or more continuous immediate upstream nodes are marked as "-".

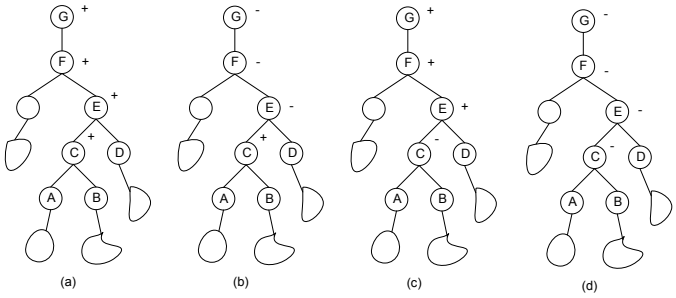


Fig. 3. Node Status Pattern

For each of the above cases, we can infer whether a node (i) has dropped packets (called *bad for sure*), (ii) is suspected to have dropped packets (called *suspiciously bad*), (ii) has not been found to drop packets (called *temporarily good*), or (iv) must have not dropped packets (called *good for sure*):

Case 1: $+\{+\}^+$. The node and its continuous immediate upstream nodes do not drop packets along the involved path, but it is unknown whether they drop packets on other forwarding paths. Therefore, the sink infers that these nodes are *temporally good*.

Case 2: $+\{-\}^+$. In the case, all nodes marked as "-" must be *bad for sure*. To show the correctness of this rule, we prove it by contradiction. Without loss of generality, we examine the scenario illustrated in Fig 3(b), where node C is marked as "+", and node E, F, G are marked as "-". If our conclusion is incorrect and node E is good, E must not drop its own packets. Since E is marked as "-", there must be some upstream nodes of E dropping E's packets. Note that the bad upstream nodes are at least one hop above E and at least two hops above C. It is impossible for it to differentiate packets from E and C, so it cannot selectively drop the packets from E while forwarding the packets from C. Even if C and the bad upstream node collude, they cannot achieve this result. This is because every packet from C must go through and be encrypted by E, and therefore the bad upstream node cannot tell the source of the packet to perform selective dropping. Note that, if a packet is forwarded to the bad upstream node without going through E, the packet cannot be correctly decrypted by the sink and thus will be dropped. Therefore, E must be bad. Similarly, we can also conclude that F and G are also bad.

Case 3: $-\{+\}^+$. In this case, either the node marked as "-" or its parent marked as "+" must be bad. But it cannot be further inferred whether (i) only the node with "-" is bad, (ii) only the node with "+" is bad, or (iii) both nodes are bad. Therefore, it is concluded that both nodes are *suspiciously bad*.

Case 4: $-\{-\}^+$. In this case, every node marked with "-" could be bad or good. Conservatively, they have to be considered as *suspiciously bad*. Specifically, suppose v is the highest-level node that is marked as "-", and u is its parent

node. If u is the sink, v must be *bad for sure*, otherwise, both u and v are *suspiciously bad*. On the other hand, suppose v is a child of u and they are both marked as "-", if the dropping rate of u is larger than that of v by at least θ (i.e., $D_v < D_u$ and $D_u - D_v > \theta$, recalling that θ is a threshold used to tolerate incidental droppings), then node u is *bad for sure*. Otherwise, both u and v are *suspiciously bad*.

C. Tree Reshaping and Three Ranking Algorithms

The tree used to forward data is dynamically changed from round to round, which enables the sink to observe the behavior of every sensor node in a large variety of routing topologies. For each of these scenarios, node categorization algorithm is applied to identify sensor nodes that are bad for sure or suspiciously bad. After multiple rounds, sink further identifies bad nodes from those are suspiciously bad by applying several proposed heuristic methods.

1) Tree Reshaping: The tree used for forwarding data from sensor nodes to the sink is dynamically changed from round to round. In other words, each sensor node may have a different parent node from round to round. To let the sink and the nodes have a consistent view of their parent nodes, the tree is reshaped as follows. At the beginning of each round i ($i = 1, 2, \cdots$), node u picks the $[h^i(K_u) \text{ MOD } n_{p,u}]^{th}$ parent node as its parent node for this round, where h is a hash function and $h^i(K_u) = h(h^{i-1}(K_u))$. Note that, how the parents are selected is predetermined by both the preloaded secret K_u and the list of parents recorded in the tree establishment phase. The selection is known by the sink. Therefore, a misbehaving node cannot arbitrarily select its parent in favor of its attacks.

2) Identifying Most Likely Bad Nodes from Suspiciously Bad Nodes: After a round ends, the sink calculates the dropping rate of each node, and runs node categorization algorithm to identify nodes that are bad for sure or suspiciously bad. Since the number of suspiciously bad nodes are potentially large, we propose how to identify most likely bad nodes from the suspiciously bad nodes as follows. By examining the rules in Case 3 and Case 4 for identifying suspiciously bad nodes, we can see that, in each of these cases (i) there are two nodes, denoted as u and v, which have the same probability to be the bad nodes and (ii) at least one of them must be bad. We call these two nodes as a *suspicious pair*. For each round i, all identified suspicious pairs are recorded in a *suspicious set* denoted as $S_i = \{\langle u_j, v_j\rangle | \langle u_j, v_j\rangle$ is a suspicious pair and $\langle u_j, v_j\rangle = \langle v_j, u_j\rangle\}$. Therefore, after n rounds of detection, we can obtain a series of suspicious sets: $S_1, S_2, \cdots, S_n$.

We define $\overline{S}$ as the set of most likely bad nodes identified from $S_1, S_2, \cdots, S_n$, if $\overline{S}$ has the following properties:

- *Coverage*. $\forall \langle u, v\rangle \in S_i$ ($i = 1, \cdots, n$), it must hold that either $u \in \overline{S}$ or $v \in \overline{S}$. That is, for any identified suspicious pair, at least one of the nodes in the pair must be in the set of most likely bad nodes.
- *Most-likeliness*. $\forall \langle u, v\rangle \in S_i$ ($i = 1, \cdots, n$), if $u \in \overline{S}$ but $v \notin \overline{S}$, then u must have higher probability to be bad than v based on n rounds of observation.
- *Minimality*. The size of $\overline{S}$ should be as small as possible in order to minimize the probability of mis-accusing innocent nodes.

Among the above three conditions, the first one and the third one can be relatively easily implemented and verified. For the second condition, we propose several heuristics to find nodes with *most-likeliness*.

Global Ranking-Based (GR) Method

The GR method is based on the heuristic that, the more times a node is identified as suspiciously bad, the more likely it is a bad node. With this method, each suspicious node u is associated with an *accused account* which keeps track of the time that the node has been identified as suspiciously bad nodes. To find out the most likely set of suspicious nodes after n rounds of detection, as described in Algorithm 1, all suspicious nodes are ranked based on the descending order of the values of their accused accounts. The node with the highest value is chosen as a most likely bad node and all the pairs that contain this node are removed from $S_1, \cdots, S_n$, resulting in new sets. The process continues on the new sets until all suspicious pairs have been removed.

Algorithm 1 The Global Ranking-Based Approach

1: Sort all suspicious nodes into queue Q according to the descending order of their accused account values
2: $\overline{S} \leftarrow \emptyset$
3: **while** $\bigcup_{i=1}^{n} S_i \neq \emptyset$ **do**
4: $\quad u \leftarrow deque(Q)$
5: $\quad \overline{S} \leftarrow \overline{S} \wedge \{u\}$
6: $\quad$ remove all $\langle u, *\rangle$ from $\bigcup_{i=1}^{n} S_i$

Stepwise Ranking-Based (SR) method

It can be anticipated that the GR method will misaccuse innocent nodes that have frequently been parents or children of bad nodes: as parents or children of bad nodes, according to previously-described rules in Cases 3 and 4, the innocents can often be classified as suspiciously bad nodes. To reduce misaccusation, we propose the SR method. With the SR method, the node with the highest accused account value is still identified as a most likely bad node. However, once a bad node u is identified, for any other node v that has been suspected together with node u, the value of node v's accused account is reduced by the times that u and v have been suspected together. This adjustment is motivated by the possibility that v has been framed by node u. After the adjustment, the node that has the highest value of accused account among the rest nodes is identified as the next mostly like bad node, which is followed by the adjustment of the accused account values for the nodes that have been suspected together with the node. Note that, similar to the GR method, after a node u is identified as bad, all suspicious pairs with format $\langle u, *\rangle$ are removed from $S_1, \cdots, S_n$. The above process continues until all suspicious pairs have been removed. The SR method is formally presented in Algorithm 2.

Hybrid Ranking-Based (HR) Method

Algorithm 2 The Stepwise Ranking-Based Approach

1: $\overline{S} \leftarrow \emptyset$
2: **while** $\bigcup_{i=1}^{n} S_i \neq \emptyset$ **do**
3: $u \leftarrow$ the node has the maximum times of presence in $S_1, \cdots, S_n$
4: $\overline{S} \leftarrow \overline{S} \wedge \{u\}$
5: remove all $\langle u, * \rangle$ from $\bigcup_{i=1}^{n} S_i$

The GR method can detect most bad nodes with some misaccusations while the SR method has fewer misaccusations but may not detect as many bad nodes as the GR method. To balance the tradeoff, we further propose the HR method, which is formally presented in Algorithm 3. According to HR, the node with the highest accused account value is still first chosen as most likely bad node. After a most likely bad node has been chosen, the one has the highest accused account value among the rest is chosen only if the node has not always been accused together with the bad nodes that have been identified already. Thus, the accusation account value is considered as an important criterion in identification, as in the GR method; meanwhile, the possibility that an innocent node being framed by bad nodes is also considered by not choosing the nodes who have always being suspected together with already-identified bad nodes, as in the SR method.

Algorithm 3 The Hybrid Ranking-Based Approach

1: Sort all suspicious nodes into queue Q according to the descending order of their accused account values
2: $\overline{S} \leftarrow \emptyset$
3: **while** $\bigcup_{i=1}^{n} S_i \neq \emptyset$ **do**
4: $u \leftarrow deque(Q)$
5: **if** there exists $\langle u, * \rangle \in \bigcup_{i=1}^{n} S_i$ **then**
6: $\overline{S} \leftarrow \overline{S} \wedge \{u\}$
7: remove all $\langle u, * \rangle$ from $\bigcup_{i=1}^{n} S_i$

D. Handling Collusion

Compromised nodes that are located close with each other may collude to render the sink to mis-accuse some innocent nodes. The attackers do not gain any benefit if the collusion triggers the scenarios of Case 1 and Case 2. However, they can mis-accuse honest nodes if the collusion triggers the scenarios of Case 3 and Case 4. As a general example shown in Fig. 4, if nodes B, C and D are compromised and collude, they will drop all or some of the packets of their own and their downstream nodes. Consequently, according to the rules in Case 3, $\langle A, B \rangle$, $\langle A, C \rangle$ and $\langle A, D \rangle$ are all identified as pairs of suspiciously bad nodes. Since A has been suspected for more times than B, C and D, it is likely that A is mis-accused as bad node. Similarly, if nodes B and E are compromised and collude, B may drop some packets of itself and its downstream nodes, and then E further drops packets from B and its downstream nodes including E and E's downstream nodes. Consequently, the dropping rates for E and its downstream nodes are higher than that for node A. According to Case 4, $\langle E, A \rangle$ and $\langle A, B \rangle$ are both identified as pairs of suspiciously bad nodes. Since A has been suspected for more times than B and$E$, it is likely to be identified as bad node.

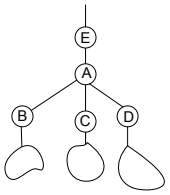


Fig. 4. Collusion Scenarios

To defeat collusion that may lead to mis-accusation, our scheme is extended as follows:

- The concept of *suspicious pair* is extended to *suspicious tuple* which is a non-ordered sequence of suspicious nodes. Note that, a suspicious pair is a special case of suspicious tuple,i.e., suspicious 2-tuple.
- A new rule is introduced: for each round i, if there exists multiple suspicious tuples of which each contains a certain node u, $\langle u, v_{1,1}, \cdots, v_{1,m_1} \rangle, \cdots, \langle u, v_{n,1}, \cdots, v_{n,m_n} \rangle$, all these tuples should be combined into a single tuple without duplication. For example, if the original tuples are $\langle u, v_1 \rangle$, $\langle u, v_2, v_3 \rangle$ and $\langle u, v_3 \rangle$, these tuples will be replaced with $\langle u, v_1, v_2, v_3 \rangle$, where each of the four nodes is suspected for only once.

As to be shown in our simulation results, the above enhancement can deal with collusions at the cost of slightly degraded detection rate.

E. An Extension for Identifying Packet Modifiers

If a compromised node modifies the packets that it is supposed to forward, the node can be detected with the afore-described scheme. This is because, modified packets will be detected by the sink and thus be dropped(detailed in step(iii) of the packet receipt procedure at sink). This is equivalent to that the packets are dropped by the modifier; hence, the packet modifier can be identified as a packet dropper. However, detecting modifiers in this way is not ideal because modified packets cannot be identified earlier by en-route nodes to save energy and bandwidth consumption. To enable en-route detection of modifications, the afore-described procedures for packet sending and forwarding can be slightly modified as follows.

When a node u has a data item D to report, it can obtain endorsement message authentication codes (MACs) from its neighbors, which are denoted as $MAC(D)$, following existing en-route filtering schemes such as the statistical en-route filtering scheme (SEF) [7] and the interleaved hop-by-hop authentication scheme [8]. The source node u generates and sends the following packet to its parent node P_u:

$$\langle P_u, D, MAC(D), \{R_u, u, C_p \text{ MOD } N_s, pad_{u,0}\}_{K_u}, pad_{u,1} \rangle.$$

When packet $\langle v, D, MAC(D), m \rangle$ is received by an en-route node v, node v can check the integrity of D in the same

way as in existing packet filtering schemes [7], [8]. If a packet is found modified, it is immediately dropped; otherwise, the following packet is forwarded by v:

$$\langle P_v, D, MAC(D), \{R_v, m'\}_{K_v} \rangle,$$

where m' is constructed the same way from m as in the scheme to identify packet droppers.

IV. Performance Evaluation

A. Objectives, Metrics, and Methodology

Our packet dropper/modifier identification scheme is implemented in the ns-2 simulator (version 2.30) to evaluate the effectiveness and efficiency of the proposed scheme. We measure the performance of our scheme from two aspects: *the detection rate*, defined as the ratio of successfully identified bad nodes, and *the false positive probability*, defined as the ratio of mis-accused innocent nodes over all innocent nodes.

We run simulations on a $400 \times 400m^2$ network with randomly generated network topology. Unless otherwise stated, we set the percentage of bad nodes to 10%, the network size to 100 sensor nodes, the per-node packet reporting interval to 3 seconds, and the length of each round to 300 seconds. Also, when a bad node decides to drop packet in a round, it drops 30% of the packets.

Attack Model: Compromised nodes might treat packets generated by themselves and those by other nodes differently. For their own packets, a compromised node may (1) drop the packets at each round, (2) drop the packets in some randomly rounds, or (3) do not drop all the time. For other nodes' packets that it is supposed to forward, a compromised node may (1) drop the packets in each round, or (2) drop the packets in some randomly rounds. Consider the combination of dropping behaviors in the above two categories, we obtain six attack models in total, namely, attack models 1-1, 1-2, 2-1, 2-2, 3-1 and 3-2, where the first index represents the dropping behavior towards the packets of the bad node itself and the second index represents the dropping behavior towards others' packets.

B. Simulation Results

We first report the simulation results when there is no node collusion and then the results when there is collusion.

1) Evaluation of Ranking Algorithms: Fig 5 shows the detection rate and false positive probability of our scheme under different attack models. From the figure, we can see that the stepwise ranking (SR) algorithm provides a bit lower detection rate than the other two ranking algorithms in the first several rounds, but after 8 rounds, the three ranking algorithms achieve almost the same detection rate. In terms of false positive probability, the global ranking (GR) algorithm introduces much higher false positive probability than the other two, while the other two algorithms result in almost the same number of false positives. This is because the global ranking (GR) algorithm identifies bad nodes only based on the number that a node is suspected. Therefore, if an innocent node does not have many choices to select its parents in different rounds, or many of its possible parent nodes are actually compromised, the times that this innocent node is suspected will be large. On the contrary, the hybrid ranking (HR) and the stepwise ranking (SR) algorithms do not select a node which is suspected many times when that node has always been suspected together with some already-identified bad nodes, which results in less number of mis-accusations. Consider both the metrics, it is determined that the hybrid ranking is the best ranking algorithm among the three for its high detection rate and low false positive.

2) Impact of the Number of Rounds: We study the number of rounds needed to collect information such that a stable and high detection rate as well as a low false positive probability is achieved. We use the hybrid ranking (HR) algorithm here and first plot the detection rate under the six attack models in each round in Fig. 6. From the figure, we can see almost all bad nodes can be identified after 8 rounds regardless of the attack model. Among them, under attack model 1-2, the bad nodes will be detected quickly after 5 rounds. This is because a bad node does not drop packets from its downstream nodes at some intervals, which results in the $+\{-\}^+$ case and the bad nodes can be identified immediately according to our proposed rule. On the contrary, under attack model 3-2, more rounds are needed to achieve a higher detection rate. In this case, bad nodes are sly and do not drop their self-generated packets. Consequently, they are only categorized as suspiciously bad nodes. More rounds are needed before they are eventually identified via a ranking algorithm.

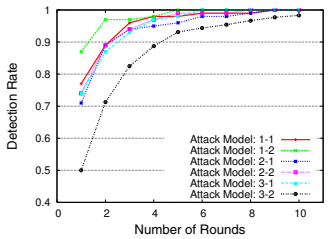


Fig. 6. Number of Rounds vs. Detection Rate

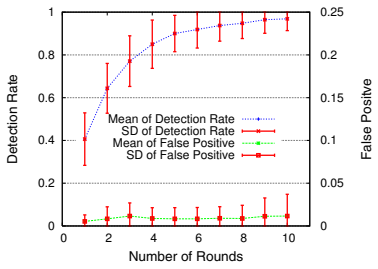
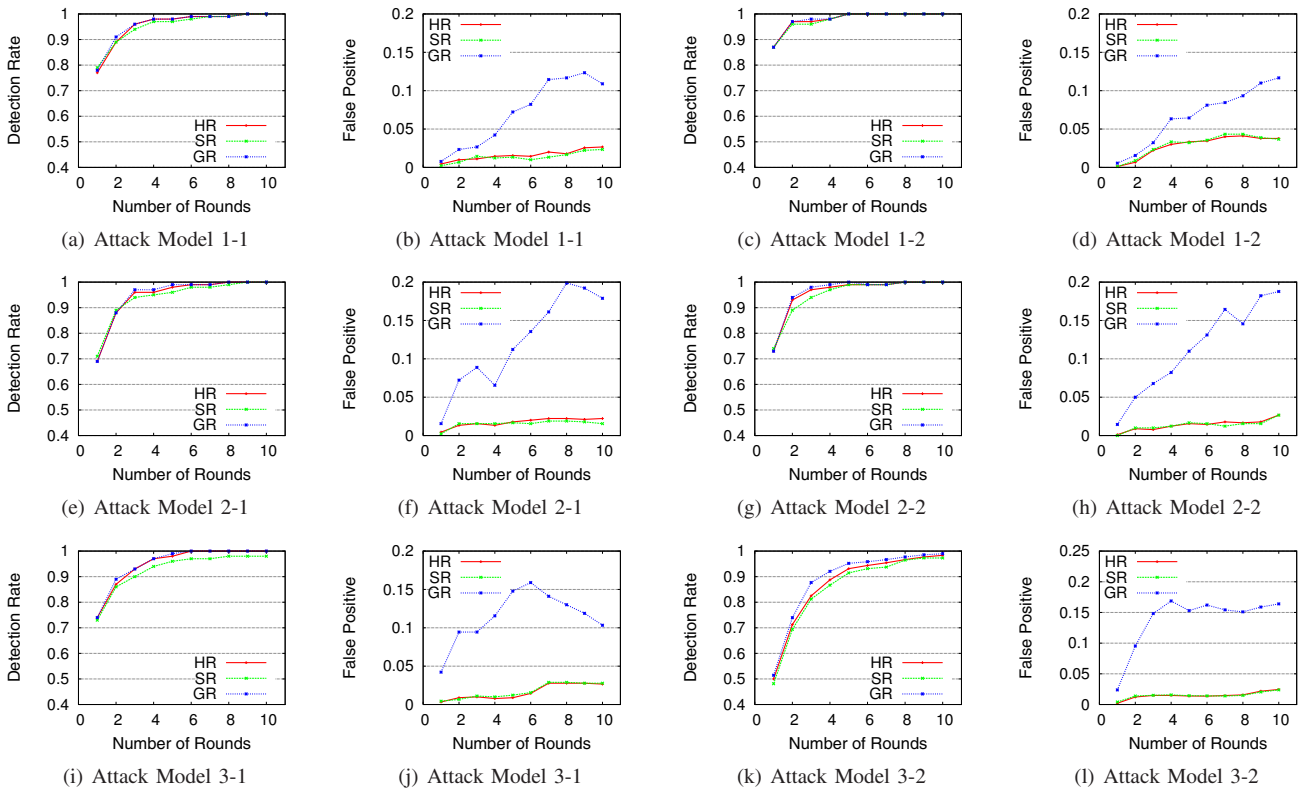


Fig. 7. Mean and Standard Deviation for the HR Method

Since the attack model 3-2 is the most difficult one, we study the standard deviations of the detection rate and the false positive probability under this attack model. The data used to compute the standard deviations are obtained from the simulations run over 50 random network topologies. The



(a) Attack Model 1-1 (b) Attack Model 1-1 (c) Attack Model 1-2 (d) Attack Model 1-2

(e) Attack Model 2-1 (f) Attack Model 2-1 (g) Attack Model 2-2 (h) Attack Model 2-2

(i) Attack Model 3-1 (j) Attack Model 3-1 (k) Attack Model 3-2 (l) Attack Model 3-2

Fig. 5. Comparing Ranking Strategy under Various Attack Models

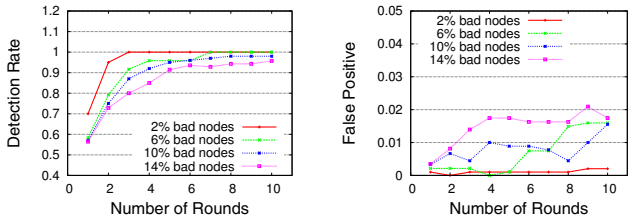
simulation results are shown in Fig. 7. As we can see, the standard deviation of detection rate becomes smaller and smaller as the number of rounds increases. It becomes stable after 8 rounds at about 0.125. The standard deviation of the false positive probability is higher than that of detection rate, but it is still as low as 0.15.

In the following experiments, we study various of impacts of system parameters based on attack model 3-2 with hybrid ranking algorithm.

3) Impact of Percentage of Bad Nodes: Fig. 8 shows the detection performance as the percentage of bad nodes varies. Generally, the less the number of bad nodes, the easier to identify these nodes. However, after a multiple rounds of identification, the detection rates under different percentage of bad nodes become similar, and all of them achieve very high detection rate.

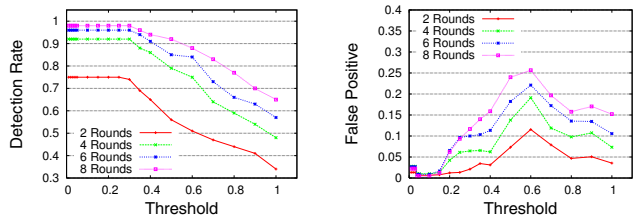
4) Impact of Thresholds: (1)Threshold for Differentiating "+" Nodes and "-" Nodes. In order to make our scheme to tolerate natural packet loss, we use a threshold θ when marking each node with "+" or "-". Fig. 9 shows the impact of this threshold on the detection performance. As depicted in Fig. 9(a), the larger is the threshold, the lower is the detection rate. This is because, less nodes will be marked as "-" as the threshold increases; hence, a part of bad nodes will escape from being detected.

As shown in Fig. 9(b), when the threshold increases, the false positive probability increases first and then decreases after the threshold reaches a certain value (turning point). Hence, we select the threshold to be 0.1, with which high detection rate and low false positive can be achieved simultaneously, as shown in Fig. 9.



(a) Detection Rate (b) False Positive

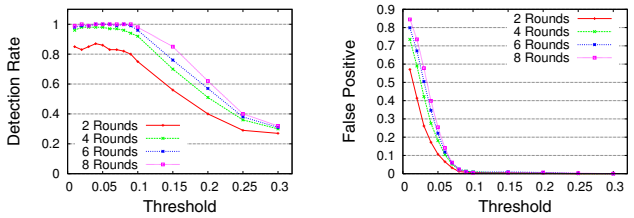
Fig. 8. Impact of Percentage of Bad Nodes



(a) Detection Rate (b) False Positive

Fig. 9. Threshold for Differentiating "+" Nodes and "-" Nodes

(2)Threshold for Identifying Nodes with Dropping Rates. Considering that incidental collisions may cause two nodes to have different dropping rates, we use a threshold to differentiate the case that two nodes really have different dropping rates with the case there are incidental collisions. Fig. 10 shows the impact of this threshold on the detection rate and the false positive. We can see that, the larger the threshold, the lower the detection rate and the false positive probability. This is because, the difference in dropping rates between two nodes is an important parameter for our ranking algorithms to differentiate the behaviors between parent-child nodes. If the threshold is too large, our algorithms cannot find the abnormal behaviors that are solely reflected on the dropping rate difference. If the threshold is too small, the false positive probability will be increased, which is shown in Fig. 10(b). Based on our simulation, we found out threshold 0.1 can render a high detection rate and low false positive probability.



(a) Detection Rate (b) False Positive

Fig. 10. Threshold for Identifying Nodes with Different Dropping Rates

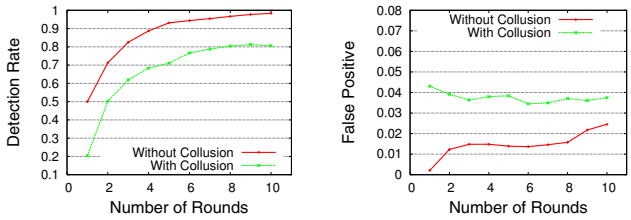
5) Impact of Node Collusion: In the collusion attack model, shown as Fig. 4, three possible collusion cases can be as follows.

Case 1: Node A, B, C and D are all compromised nodes: every node behaves normally without dropping their own packets and forwarding packets. In this attack model, these compromised nodes collude to protect themselves.

Case 2: Node A is a good node and more than one of its child nodes is compromised: the compromised nodes drop their own packets and/or packets from their own children with the same dropping rate. In this attack model, these compromised nodes collude to frame the parent node A.

Case 3: Node A is a good node but nodes B and E are not. Both bad nodes B and E drop packets of their own and/or from their downstream nodes. As a result, the dropping rates of B and its downstream nodes are the same, those of B and A are different, and those of B and E are also different.

We compare the detection rates under the collusion scenarios and the non-collusion scenarios and show the results in Fig. 11. The hybrid ranking algorithm is used. We can see that, the detection performance degrades under the collusion scenarios. But the detection rate is still as high as 80% and a low false positive probability is maintained. The reason for lower detection rate can be explained as follows: When there are collusion, multiple colluding bad nodes and one or more innocent nodes are put into a single tuple. However, if there is no collusion, generally there is only one bad node in a tuple. Hence, the bad nodes' overall times of being suspected is reduced when there is collusion, which degrades the efficiency of identifying bad nodes. In fact, if a set of bad nodes collude together most of time, only one of these nodes can be identified.



(a) Detection Rate (b) False Positive

Fig. 11. Comparison between collusion and non-collusion

V. Conclusion

To address the problem of packet dropping and modification, we proposed a simple yet effective scheme to identify misbehaving forwarders that drop or modify packets. Extensive analysis and simulations have been conducted and verified the effectiveness of the proposed scheme in various scenarios.

Acknowledgements

This work is partially supported by NSF CNS-0716744, CNS-0627354, CNS-0831906, CNS-0834585 and ONR N000140910748

References

[1] H.Chan and A. Perrig, "Security and Privacy in Sensor Networks," *IEEE Computer*, October 2003.

[2] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: attacks and countermeasures," *the First IEEE International Workshop on Sensor Network Protocols and Applications*, pp. 113–127, May 2003.

[3] V. Bhuse, A. Gupta, and L. Lilien, "Dpdsn: Detection of packet-dropping attacks for wireless sensor networks," *In the Trusted Internet Workshop, International Conference on High Performance Computing*, December 2005.

[4] S. Marti, T. Giuli, K. Lai, and M. Baker, "Mitigating Routing Misbehavior in Mobile Ad Hoc Networks," *ACM MobiCom*, August 2000.

[5] R. Roman, J. Zhou, and J. Lopez, "Applying intrusion detection systems to wireless sensor networks," *Third IEEE Annual Consumer Communications and Networking Conference (CCNC)*, pp. 640–644, Jan. 2006.

[6] S. Lee and Y. Choi, "A resilient packet-forwarding scheme against maliciously packet-dropping nodes in sensor networks," *Proceedings of the fourth ACM workshop on Security of ad hoc and sensor networks (SASN)*, pp. 59–70, 2006.

[7] F. Ye, H. Luo, S. Lu, and L. Zhang, "Statistical En-route Filtering of Injected False Data in Sensor Networks," *IEEE INFOCOM*, March 2004.

[8] S. Zhu, S. Setia, S. Jajodia, and P. Ning, "An Interleaved Hop-by-Hop Authentication Scheme for Filtering False Data in Sensor Networks," *IEEE Symposium on Security and Privacy*, 2004.

[9] H. Yang, F. Ye, Y. Yuan, S. Lu, and W. Arbaugh, "Toward Resilient Security in Wireless Sensor Networks," *ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, May 2005.

[10] Z. Yu and Y. Guan, "A Dynamic En-route Scheme for Filtering False Data in Wireless Sensor Networks," *IEEE Infocom 2006*, April 2006.

[11] F. Ye, H. Yang, and Z. Liu, "Catching Moles in Sensor Networks," *IEEE International Conference on Distributed Computing Systems (ICDCS)*, June 2007.

[12] Q. Li and D. Rus, "Global clock synchronization in sensor networks," *IEEE INFOCOM*, 2004.