Catching Packet Droppers and Modifiers in Wireless Sensor Networks

Chuang Wang, Taiming Feng, Jinsook Kim, Guiling Wang, and Wensheng Zhang

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 or tolerate such attacks but very few can effectively and efficiently identify the intruders. To address this problem, we propose a simple yet effective scheme, which can identify misbehaving forwarders that drop or modify packets. Extensive analysis and simulations have been conducted to verify the effectiveness and efficiency of the scheme.

Index Terms

Packet Dropping, Packet Modification, Intrusion Detection, Wireless Sensor Networks.

- *C. Wang, T. Feng, J. Kim and W. Zhang are with the Department of Computer Science, Iowa State University, Ames, IA, 50010.E-mail: {cwang,taiming,dvorakjs,wzhang}@cs.iastate.edu*
- *G. Wang is with the Department of Computer Science, New Jersey Institute of Technology, Newark, NJ, 07102. E-mail: gwang@njit.edu*

Catching Packet Droppers and Modifiers in Wireless Sensor Networks

1 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. Because of the ease of deployment, the low cost of sensor nodes and the capability of self-organization, 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], [4], [5], 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. To deal with packet modifiers, most of existing countermeasures [6], [7], [8], [9] aim to filter modified messages en-route within a certain number of hops. These countermeasures can tolerate or mitigate the packet dropping and modification attacks, but the intruders are still there and can continue attacking the network without being caught.

To locate and identify packet droppers and modifiers, it has been proposed that nodes continuously monitor the forwarding behaviors of their neighbors [10], [11], [12], [13], [14], [15] to determine if their neighbors are misbehaving, and the approach can be extended by using the reputation-based mechanisms to allow nodes to infer whether a non-neighbor node is trustable [16], [17], [18], [19]. This methodology may be subject to high energy cost incurred by the promiscuous operating mode of wireless interface; moreover, the reputation mechanisms have to be exercised with cautions to avoid or mitigate bad mouth attacks and others. Recently, Ye *et al.* proposed a probabilistic nested marking (PNM) scheme [20]. But with the PNM scheme, modified packets should not be filtered out en-route because they should be used as evidence to infer packet modifiers; hence, it cannot be used together with existing packet filtering schemes.

In this paper, we propose a simple yet effective scheme to catch both packet droppers and modifiers. In this scheme, a 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 ratio associated with every sensor node, and then runs 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 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 runs 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.

Our proposed scheme has the following features: (i) being effective in identifying both packet droppers and modifiers, (ii) low communication and energy overheads, and (iii) being compatible with existing false packet filtering schemes; that is, it can be deployed together with the false packet filtering schemes, and therefore it can not only identify intruders but also filter modified packets immediately after the modification is detected. Extensive simulation on ns2 simulator has been conducted to verify the effectiveness and efficiency of the proposed scheme in various scenarios.

In the rest of the paper, Section 2 defines the system model. Section 3 describes the proposed scheme and section 4 reports the evaluation results. Section 5 discusses the related work, and Section 6 concludes the paper.

2 System Model

2.1 Network Assumptions

We consider a typical deployment of sensor networks, where a large number of sensor nodes are randomly deployed in a two dimensional area. Each sensor node generates sensory data periodically and all these nodes collaborate to forward packets containing the data towards a sink. The sink is located within the network. We assume all sensor nodes and the sink are loosely time synchronized [21], which is required by many applications. Attack-resilient time synchronization schemes, which have been widely investigated in wireless sensor networks [22], [23], can be employed. The sink is aware of the network topology, which can be achieved by requiring nodes to report their neighboring nodes right after deployment.

2.2 Security Assumptions and Attack Model

We assume the network sink is trustworthy and free of compromise, and the adversary cannot successfully compromise regular sensor nodes during the short topology establishment phase after the network is deployed. This assumption has been widely made in existing work [8], [24]. After then, the 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 framing 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.

3 The Proposed Scheme

Our proposed scheme consists of a system initialization phase and several equal-duration rounds of intruder identification phases.

- In the initialization phase, sensor nodes form a topology which is a directed acyclic graph (DAG). A routing tree is extracted from the DAG. Data reports follow the routing tree structure.
- In each round, data is transferred through the routing tree to the sink. 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 a node categorization algorithm to identify nodes that must be bad (i.e., packet droppers or modifiers) and nodes that are suspiciously bad (i.e., suspected to be packet droppers and modifiers).
- 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 DAG 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.

3.1 DAG Establishment and Packet Transmission

All sensor nodes form a DAG and extracts a routing tree from the DAG. The sink knows the DAG and the routing tree, and shares a unique key with each node. When a node wants to send out a packet, it attaches to the packet a sequence number, encrypts the packet only with the key shared with the sink, and then forwards the packet to its parent on the routing tree. 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 tracks 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 ratio for every node. Based on the dropping ratio and the knowledge of the 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.

3.1.1 System Initialization

The purpose of system initialization is to set up secret pairwise keys between the sink and every regular sensor node, and to establish the DAG and the routing tree to facilitate packet forwarding from every sensor node to the sink.

3.1.1.1 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.
- L_r: the duration of a round.
- N_p: the maximum number of parent nodes that each node records during the DAG 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.

3.1.1.2 Topology 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 on the DAG if it has received $\langle w, d_w\rangle$ where $d_w = d_v$. That is, node u records as its parents on the DAG the nodes whose distance (in hops) to the sink is the same and the distance is one hop shorter than its own. If the number of such parents is greater than N_p, only N_p parents are recorded while 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 let its downstream one-hop neighbors to continue the process of DAG establishment. Then, among the recorded parents on the DAG, node u randomly picks one (whose ID is denoted as P_u) as its parent on the routing tree. Node u also picks a random number (which is denoted as R_u) between 0 and $N_p - 1$. As to be elaborated later, random number R_u is used as a short ID of node u to be attached to each packet node u forwards, so that the sink can trace out the forwarding path. Finally,

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.

node u sends P_u, R_u and all recorded parents on the DAG to the sink.

After the above procedure completes, a DAG and a routing tree rooted at the sink is established. The routing tree is used by the nodes to forward sensory data until the tree changes later; when the tree needs to be changed, the new structure is still extracted from the DAG.

The lifetime of the network is divided into rounds, and each round has a time length of L_r. After the sink has received the parent lists from all sensor nodes, it sends out a message to announce the start of the first round, and the message is forwarded hop by hop to all nodes in the network. Note that, each sensor node sends and forwards data via a routing tree which is implicitly agreed with the sink in each round, and the routing tree changes in each round via our tree reshaping algorithm presented in Sec. 3.3.

3.1.2 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 ($0 \leq R_u \leq N_p - 1$) is a random number picked by node u during the system initialization phase, and R_u is attached to the packet to enable the sink to find out the path along which the packet is forwarded. $\{X\}_Y$ represents the result of encrypting X using key Y.

Paddings $pad_{u,0}$ and $pad_{u,1}$ are added to make all packets equal in 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 paddings are 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 off.
- 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 $\log(N_s)$ bits long. Setting $pad_{u,0}$ to this value ensures that all packets in the network have the same length 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. Suppose on a routing tree, node u is the parent of node v and v is a parent of node w. When u receives a packet from v, it cannot differentiate whether the packet is originally sent by v or w unless nodes u and v collude. Hence, the above packet sending and forwarding scheme results in the difficulty to launch selective dropping, which is leveraged in locating packet droppers. We take special consideration for the collusion scenarios, which are to be elaborated later.

3.1.3 Packet Receiving 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:

(i) Two temporary variables u and m are introduced. Let $u = 0$ and $m = m'$ initially.

(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 for all children nodes of node u, the packet is identified as have been 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.

The process of packet receipt at the sink can be formalized as Algorithm 1

Algorithm 1 Packet Receipt at the Sink

```
1: Input: packet ⟨0, m⟩.
2: u = 0, m' = m;
3: hasSuccAttemp = false;
4: for each child node v of node u do
5:    P=dec(K_v, m');
6:    if decryption fails then
7:       continue;
8:    else
9:       hasSuccAttemp = true;
10:      if P starts with ⟨R_v, v⟩ then
11:         record the sequence number; /* v is the sender */
12:         break;
13:      else
14:         trim R_v from P and get m'; /*v is a forwarder */
15:         u ← v, hasSuccAttemp = false; go to line 4;
16: if hasSuccAttemp = false then
17:    drop this packet;
```

3.1.4 An Example

Fig 1 shows an example sensor network with 7 nodes, nodes $0-6$. Node ID is represented by 3 bits. Suppose the maximum

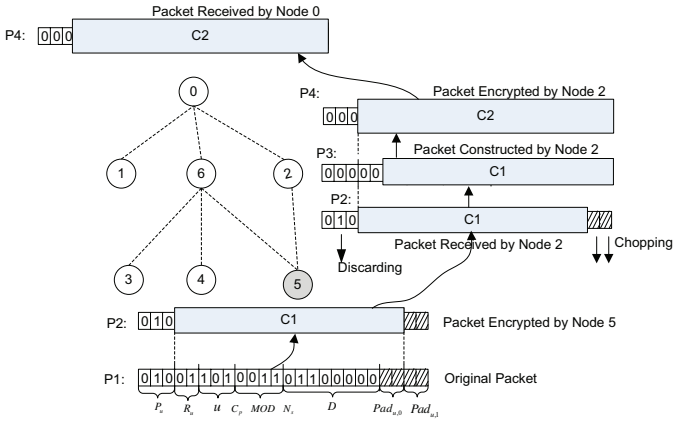


Fig. 1. Example of Packet Sending, Forwarding

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 following procedure: node 5, which is 2 hops away from the sink, generates sensory data 96; the data is sent to the sink node 0. Assume 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, which 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 off the rightmost $\log N_p$ bits, which are 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 rightmost 2 bits are chopped off, and a random number R_2 with 2 bits 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 start 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.

3.2 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 ratio 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:

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

Based on the dropping ratio 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 ratio 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 then, for each path from a leaf node to the sink, the nodes' mark pattern in this path can be decomposed into any combination of the following basic patterns, which are also illustrated by Fig. 2:

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

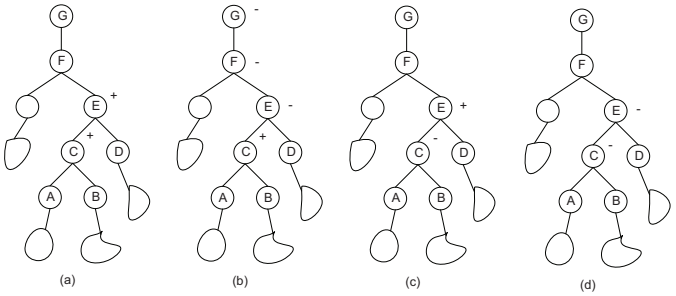


Fig. 2. Node Status Pattern

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*), (iii) 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 parent node 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 *temporarily good*. For example,

in Fig. 2(a), node C and E are marked "+" and are regarded as temporarily good. A special case is, if a leaf node is marked as "+", it is safe to infer it is good since it cannot drop other's packets.

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 2(b), where node C is marked as "+", and nodes E, F and 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, i.e., at least two hops above C. It is impossible for them to differentiate packets from E and C, so they 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. 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*. The correctness of this rule can also be proved by contradiction. Without loss of generality, let us consider the scenario shown in Fig. 2(c), where node C is marked as "-", and node E is marked as "+". Now suppose both C and E are good, and hence there must exist at least one upstream node of E which is a bad node that drops the packets sent by C. However, it is impossible to find such an upstream node since nodes F and G, and other upstream nodes cannot selectively drop packets from node C while forwarding packets from node E. Hence, either node C is bad or node E is bad in this case.

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 ratio 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), node u is *bad for sure*. Otherwise, both u and v are *suspiciously bad*.

Based on the above rules, we develop a node categorization algorithm to find nodes that are *bad for sure* or *suspiciously bad*. The formal algorithm is presented in Algorithm 2.

Algorithm 2 Tree-Based Node Categorization Algorithm

```
Input: Tree T, with each node u marked by "+" or "-", and its dropping ratio d_u.
for each leaf node u in T do
    v ← u's parent;
    while u is not the Sink do
        if u.mark = "+" then
            if v.mark = "-" then
                b ← v;
                repeat
                    e ← v;
                    v ← v's parents node;
                until v.mark = "+" or v is Sink
                Set nodes from b to e as bad for sure;
        else
            if v is Sink then
                Set u as bad for sure;
            if v.mark = "+" then
                if v is not bad for sure then
                    Set u and v as suspiciously bad;
            else
                if d_v − d_u > θ then
                    Set v as bad for sure;
                else if d_u − d_v > θ then
                    Set u and v as suspiciously bad;
        u ← v, v ← v's parents node
```

3.3 Tree Reshaping and 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 that are suspiciously bad by applying several proposed heuristic methods.

3.3.1 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. Suppose each sensor node u is preloaded with a hash function $h(.)$ and a secret number K_u which is exclusively shared with the sink. At the beginning of each round i ($i = 1, 2, \cdots$), node u picks the $[h^i(K_u)$ MOD $n_{p,u}]^{th}$ parent node as its parent node for this round, where $h^i(K_u) = h(h^{i-1}(K_u))$ and $n_{p,u}$ is the number of candidate parent nodes that node u recorded during the tree establishment phase. Recall that node u's candidate parent nodes are those which are one hop closer to the sink and within node u's communication range. Therefore, if node u choose node w as its parent in a round, node w will not select node u as its parent, and the routing loop will not occur. 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 implicitly agreed between each node and the sink. Therefore, a misbehaving node cannot arbitrarily select its parent in favor of its attacks.

3.3.2 Identifying Most Likely Bad Nodes from Suspiciously Bad Nodes

We rank the suspiciously bad nodes based on their probabilities of being bad, and identify part of them as most likely bad nodes. Specifically, after a round ends, the sink calculates the dropping ratio of each node, and runs the node categorization algorithm specified as Algorithm 2 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 observe that in each of these cases, there are two nodes having the same probability to be bad and 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 times 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 3, 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. The GR method is formally presented in Algorithm 3.

Stepwise Ranking-Based (SR) method

It can be anticipated that the GR method will falsely accuse 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 false accusation, 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 4.

Algorithm 3 The Global Ranking-Based Approach

```
Sort all suspicious nodes into queue Q according to the descending order of their accused account values
S̄ ← ∅
while ⋃_{i=1}^{n} S_i ≠ ∅ do
    u ← deque(Q)
    S̄ ← S̄ ∧ {u}
    remove all ⟨u, *⟩ from ⋃_{i=1}^{n} S_i
```

Algorithm 4 The Stepwise Ranking-Based Approach

```
S̄ ← ∅
while ⋃_{i=1}^{n} S_i ≠ ∅ do
    u ← the node has the maximum times of presence in S_1, ⋯, S_n
    S̄ ← S̄ ∧ {u}
    remove all ⟨u, *⟩ from ⋃_{i=1}^{n} S_i
```

Hybrid Ranking-Based (HR) Method

The GR method can detect most bad nodes with some false accusations while the SR method has fewer false accusations but may not detect as many bad nodes as the GR method. To strike a balance, we further propose the HR method, which is formally presented in Algorithm 5. 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 with 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. The HR method is formally presented in Algorithm 5.

3.4 Handling Collusion

Because of the deliberate hop by hop packet padding and encryption, the packets are not distinguishable to the upstream

Algorithm 5 The Hybrid Ranking-Based Approach

```
Sort all suspicious nodes into queue Q according to the descending order of their accused account values
S̄ ← ∅
while ⋃_{i=1}^{n} S_i ≠ ∅ do
    u ← deque(Q)
    if there exists ⟨u, ∗⟩ ∈ ⋃_{i=1}^{n} S_i then
        S̄ ← S̄ ∧ {u}
        remove all ⟨u, ∗⟩ from ⋃_{i=1}^{n} S_i
```

compromised nodes as long as they have been forwarded by an innocent node. The capability of launching collusion attacks is thus limited by the scheme. However, compromised nodes that are located close with each other may collude to render the sink to accuse some innocent nodes. We discuss the possible collusion scenarios in this section and propose strategies to mitigate the effects of collusion.

As the four cases described in section 3.2, the attackers do not gain any benefit if the collusion triggers the scenarios of Case 1 and Case 2. However, the attackers may accuse honest nodes if the collusion triggers the scenarios of Case 3 and Case 4. By exploiting the rules used by the node categorization algorithm and rank algorithm, there are two possible collusion strategies to make the sink accuse innocent nodes. We use Fig. 3 as a general example to discuss the collusion scenarios.

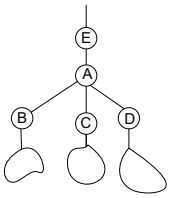


Fig. 3. Collusion Scenarios

- Horizontal Collusion: 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 falsely identified as bad node.
- Vertical Collusion: 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 its downstream nodes including B and B's downstream nodes. Note that, E cannot differentiate the packets forwarding/generating by B since they are encrypted by node A. Consequently, the dropping rates for B 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 a bad node.

To defeat collusion that may lead to false 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 collusion at the cost of slightly degraded detection rate.

3.5 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 receiving procedure at sink). This is equivalent to the case 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) [6] and the interleaved hop-by-hop authentication scheme [7]. 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 [6], [7]. 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.

Therefore, by integrating with the existing schemes [6], [7], the modified packets will be dropped by honest nodes on the way to the sink. Modified packets dropped by honest nodes are equivalent to packets dropped at the modifier nodes, which can be explained by Fig.4. Suppose node A is a compromised node and it modifies the packets it forwards randomly due to our deliberate packet encryption and padding techniques. Suppose nodes C, B and D as well as the downstream nodes of B and D are honest nodes. Node E detects that some

packets that it receives have been modified, and therefore node E drops the modified packets. Since honest nodes only drop modified packets and forward unmodified packets correctly, dropping the modified packets only affects the marks of nodes whose packets pass through the modifiers. In this example, only the marks (i.e., "+" or "-") of node A and its downstream nodes are affected by node E's dropping behavior. Therefore, modified packets dropped by honest nodes are equivalent to packets dropped at the modifier nodes in terms of the marks of each node. The sink does not need to differentiate honest nodes' behavior of dropping modified packets and compromised nodes' behavior of dropping correct packets.

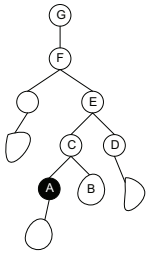


Fig. 4. Detect Packet Modifiers

4 Performance Evaluation

4.1 Objectives, Metrics, and Methodology

Our packet dropper/modifier identification scheme is simulated in the ns-2 simulator (version 2.30) to evaluate the effectiveness and efficiency of the proposed scheme. The objectives of this evaluation study are four-fold: firstly, testing the effectiveness and efficiency of our scheme in identifying packet droppers and modifiers; secondly, studying the impacts of various system parameters (i.e., sensor data reporting interval, round length, percentage of bad nodes, network scale, presence of node collusions, etc.) on the performance of our scheme; thirdly, testing the effectiveness of our scheme under six different attack models; finally, comparing the proposed global ranking (GR), stepwise ranking (SR), and hybrid ranking (HR) algorithms to provide insights on how to choose the ranking algorithm for different situations.

We measure the performance of our scheme with two metrics: *the detection rate* defined as the ratio of successfully identified bad nodes; *the false positive probability* defined as the ratio of mis-accused innocent nodes over all innocent nodes.

We run simulations in a $400 \times 400m^2$ network with randomly generated network topology. Unless stated otherwise, 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. All the results are measured and averaged based on simulations over 50 random networks.

Attack Model: We assume smart attackers selectively compromise non-leaf nodes, because compromised non-leaf nodes can attack the system more effectively than compromised leaf nodes. Compromised nodes may 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 the packets 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. For example, attack model 1-2 means that own packets are dropped at each round, while packets of others are dropped at some selected rounds. This is the easiest to identify because such attacks will result in the case of $+ - \{-\}^*$, from which the bad node can be immediately identified using our node categorization algorithm. On the other hand, attack model 3-2 means that own packets are not dropped but packets from others are dropped at some selected rounds, and experimental results demonstrate that this attack model is the hardest to deal with.

To simulate the attack behaviors in ns-2 simulator, we mimic the attack behaviors by letting each compromised node drop packets based on a particular attacking model described above. The compromised nodes are randomly selected beforehand. After initializing the simulator, each node in the network has a flag indicating whether it is compromised or not. If a node is compromised, it will mimic a certain attack behavior; otherwise, it honestly forwards or sends packets.

In the following subsections, we evaluate the proposed scheme in Sec. 4.2, and compare the proposed scheme with the PNM scheme in Sec. 4.3. The implementation of the proposed scheme on TelosB motes is reported in Sec. 4.4.

4.2 Simulation Results

4.2.1 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 achieves 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 have almost the same number of false positives. This is because the GR algorithm identifies bad nodes only based on the times 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 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 false accusations. Considering both the metrics, the HR is the best

ranking algorithm among the three for its high detection rate and low false positive.

4.2.2 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 can be achieved. We use the HR algorithm and plot the detection rate under the six attack models in Fig. 6. From the figure, we can see that 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.

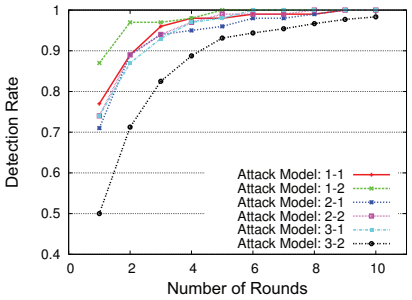


Fig. 6. Number of Rounds vs. Detection Rate

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 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.

Based on the previous experimental study, attack model 3-2 is the most effective, which renders great challenges to our proposed scheme. Also, the HR algorithm has the best performance under all attack models. Therefore, in the following, we study the impacts of various system parameters with attack model 3-2 and the HR algorithm.

4.2.3 Impact of Reporting Interval

Given a fixed time length of a round, the longer is the report interval, the less packets are sent out. When a bad node blindly drops the forwarding packets, it drops the packets from all its downstream nodes randomly and hence the percentages of its downstream nodes' packets it drops should be similar.

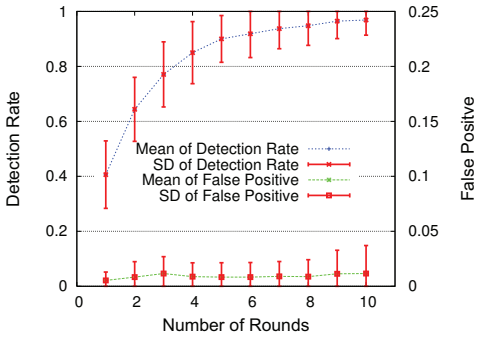
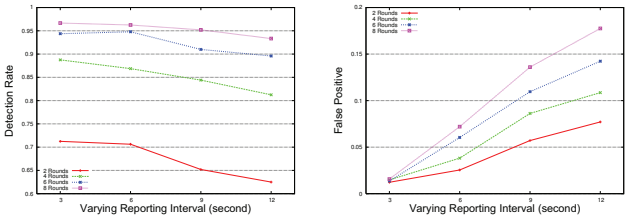


Fig. 7. Mean and Standard Deviation for the HR Method

However, when the sample space is small because of large reporting interval, the variance of the dropping ratio could be large, resulting in large false positive probability. This explains the phenomenon shown in Fig. 8(b), the false positive probability goes up when the reporting interval increases. When the number of rounds is small, Fig. 8(a) shows that the detection rate decreases as the reporting interval increases. This is because the fewer packets are sent, the less information is collected for the proposed algorithm to analyze. However, as the number of detection rounds increases, the detection rate will approach 100% regardless of reporting interval.



(a) Detection Rate (b) False Positive

Fig. 8. Impact of Reporting Interval

4.2.4 Impact of Round Length

Considering the delay for transmitting a packet from a source node to the sink, the round length affects the number of packets received at the sink in each round, which in turn affects the detection performance. Fig. 9 shows the relation between round length and the performance. It can be seen that round length mainly affects the false positive probability. As shown by Fig. 9(b), when the length is 150 seconds, the false positive probability becomes high though the detection rate is similar under different round lengths. This is because when the length of a round is small, there are not enough packets being generated and sent to the sink and the number of packets sent by different downstream nodes may not be dropped at the similar level. For example, when a bad node drops its forwarding packets, it is supposed to randomly drop the packets from all its downstream nodes. If the number of packets is small, it may drop more packets from some downstream node than others. In this situation, statistical analysis is not accurate enough and may cause relatively high

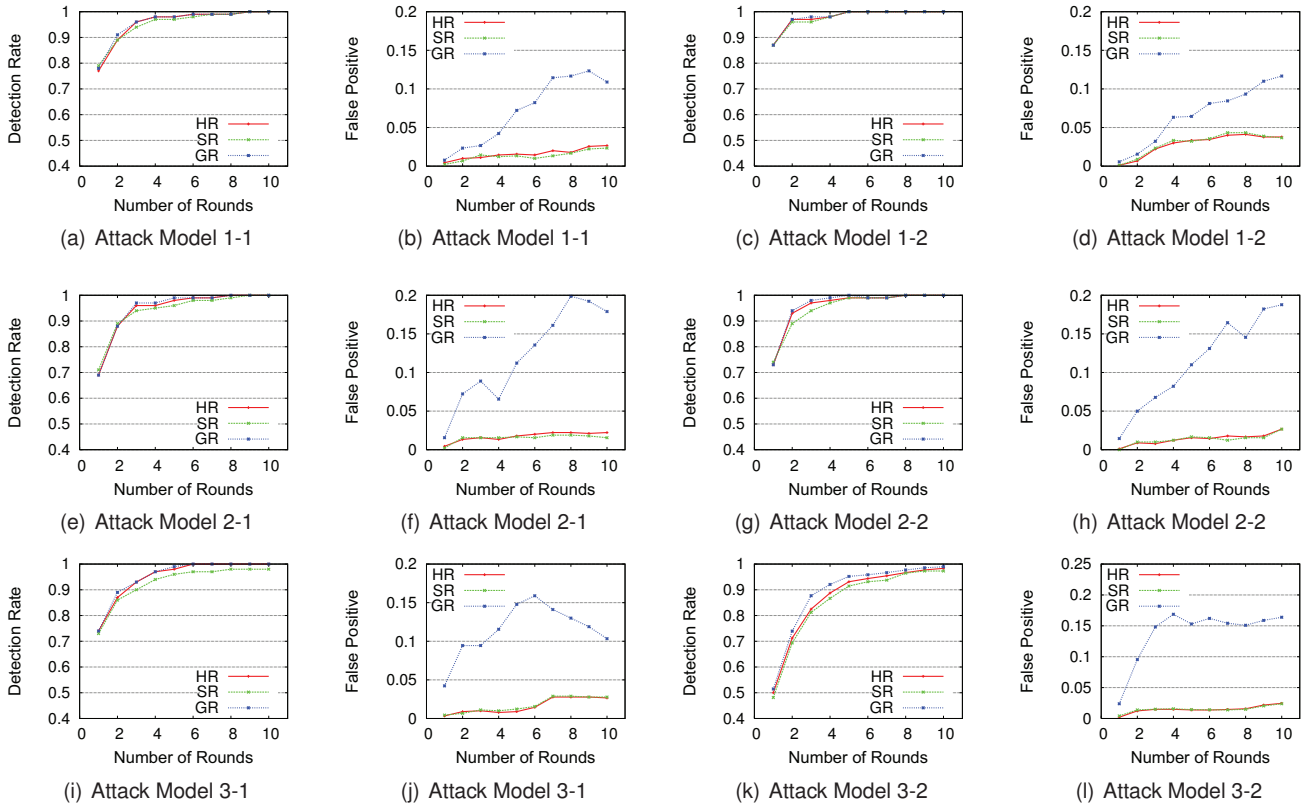


Fig. 5. Comparing Ranking Strategy under Various Attack Models

false positive. As for the detection rate, it is not sensitive to round length.

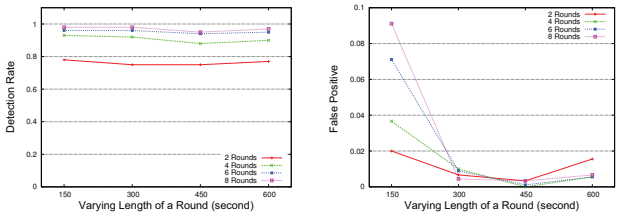


Fig. 9. Impact of Round Length

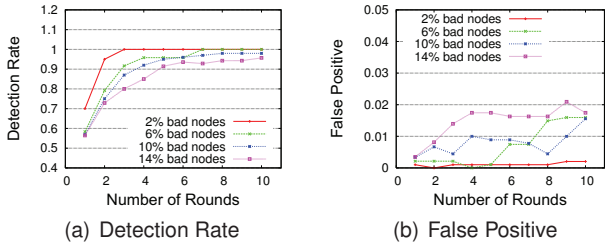


Fig. 10. Impact of Percentage of Bad Nodes

4.2.5 *Impact of Percentage of Bad Nodes*

Fig. 10 shows the detection performance as the percentage of bad nodes varies. Generally, the less is the number of bad nodes, the easier is it to identify these nodes. However, after a multiple rounds, the detection rates under different percentage of bad nodes become similar, and all of them achieve very high values.

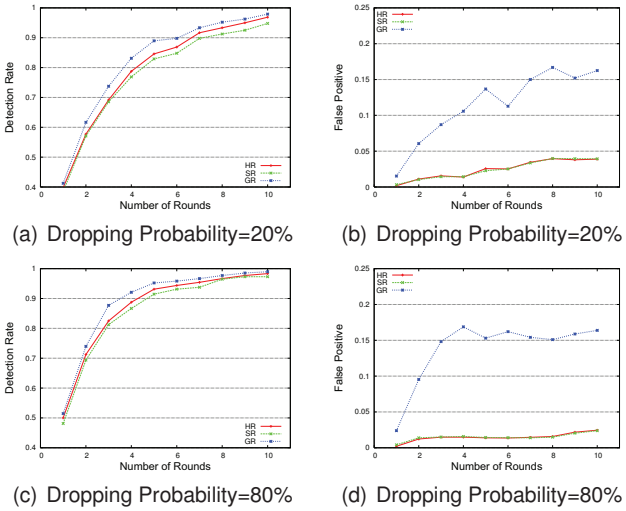
4.2.6 *Impact of Dropping Probability*

Fig. 11 shows the performance sensitivity to bad nodes' dropping percentage (i.e., the percentage of packets that will be dropped if a bad node decides to drop packets in a round). We vary the dropping probability between 20% and 80%. From Fig. 11, we can see the all the three ranking algorithms have similar sensitivity to the dropping probability. In addition, with a high dropping probability, all the three algorithms achieve a higher detection rate in the early rounds, which means they can detect bad nodes quicker, and can achieve a lower false positive generally. This is because frequent misbehaviors can quickly distinguish bad nodes from innocent nodes.

4.2.7 *Impact of Thresholds*

(1)Threshold for Differentiating "+" Nodes and "-" Nodes. In order to tolerate incidental packet loss, we use a threshold θ when marking each node with "+" or "-". Fig. 12 shows the impact of this threshold on the detection performance. As depicted in Fig. 12(a), the larger is the threshold, the lower is the detection rate. This is because, fewer nodes will be marked as "-" as the threshold increases; hence, a part of bad nodes may escape from being detected.

As shown in Fig. 12(b), when the threshold increases, the false positive probability increases first and then decreases

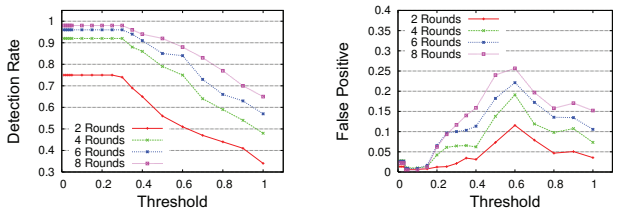


(a) Dropping Probability=20% (b) Dropping Probability=20%

(c) Dropping Probability=80% (d) Dropping Probability=80%

Fig. 11. Comparing Ranking Strategy under Various Dropping Probability

after the threshold reaches a certain value (turning point).



(a) Detection Rate (b) False Positive

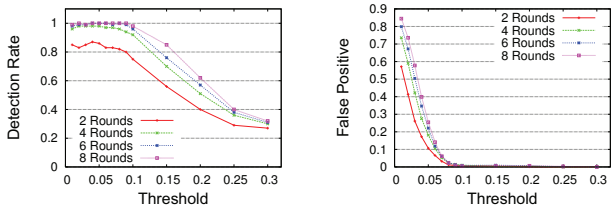
Fig. 12. Threshold for Differentiating "+" Nodes and "-" Nodes

(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 ratios from the case that the difference in their dropping ratios are caused by incidental packet loss. Fig. 13 shows the impact of this threshold on the detection rate and the false positive. We can see that, the larger is the threshold, the lower are the detection rate and the false positive probability. This is because, the difference in dropping ratios 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 ratio difference. If the threshold is too small, the false positive probability will be increased, as shown in Fig. 13(b).

4.2.8 Impact of Node Collusion

We study the following three collusion cases shown in Fig. 3.

Case 1: Nodes 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.



(a) Detection Rate (b) False Positive

Fig. 13. Threshold for Identifying Nodes with Different Dropping Rates

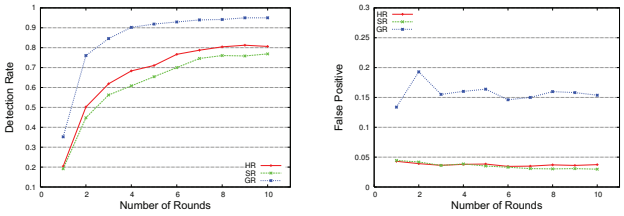
Case 2: Node A is a good node and more than one of its child nodes are compromised: the compromised nodes drop their own packets and/or packets from their children. In this case, these compromised nodes collude to frame the parent node A.

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. In this case, compromised nodes B and E collude to frame node A

We randomly generate the above collusion scenarios and conduct a set of simulations to study the impact of the extended rules discussed in Sec. 3.4

As shown by Fig. 14, under collusion attacks, the GR algorithm still has the highest detection rate and the largest false positive. The performance of SR and HR are similar to each other. And after we run our proposed scheme for 10 rounds, the detection rate and the false positive probability tend to be stable. The false positive probability of the GR algorithm has a noticeable increase from round 1 to round 2, then it goes down and becomes stable This is because, the information about suspicious nodes obtained from round 1 and round 2 is very limited, and the difference between the suspected times of bad nodes and those of innocent nodes is not big enough, which causes the increase of false positive when the GR is adopted. However, after more rounds, the accumulative suspected times of bad nodes becomes larger and larger, and the accumulative suspected times of innocent nodes increase much slower than those of bad nodes. Note that, each node randomly chooses its parents in a round based on the mutual agreement with the sink, an innocent node may choose a parent which is bad at a round, and choose innocent parents at some other rounds; hence, the accumulative times of being suspected for innocent nodes are generally fewer than those for bad nodes. In summary, the trend observed in the collusion scenarios is similar to that in the non-collusion scenarios.

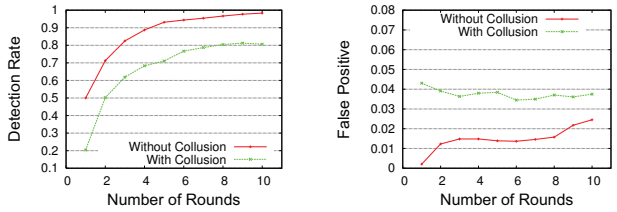
We also compare the detection rates under the collusion scenarios and the non-collusion scenarios, the results of which are shown in Fig. 15. The HR algorithm is used. We can see that, the detection performance degrades under the collusion scenarios. But the detection rate is still as high as 80% and a low false positive probability is maintained. The reason for lower detection rate can be explained as follows: When there are collusions, 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



(a) Detection Rate (b) False Positive

Fig. 14. Comparison of Ranking Strategy with Collusion

tuple. Hence, the bad nodes' overall times of being suspected is reduced when there are collusions, which degrades the efficiency in identifying bad nodes. In fact, if a set of bad nodes collude together most of the time, only one of these nodes may be identified. To deal with this issue, after one bad node is identified, it should be removed from the system and thus it cannot protect other bad nodes from being identified through continuously colluding with them.



(a) Detection Rate (b) False Positive

Fig. 15. Comparison between collusion and non-collusion

4.3 Performance Comparison

To identify packet modifiers and droppers, it has been proposed to add nested MACs to address this problem [20], [25]. Next, we compare our proposed scheme with the PNM scheme [20] in terms of detection performance and communication overhead.

4.3.1 Detection Rate and False Positive

Fig. 16 shows the overall detection performance of the PNM scheme [20] based on the same network topology and configuration of compromised nodes as in Fig. 7. The highest detection rate it can achieve is about 75%, and the lowest false positive is around 10%. However, as shown by Fig. 7, our proposed scheme can achieve 96% detection rate meanwhile maintain the false positive as low as 1%. This is because the nested MAC approach can only identify the problematic links, i.e., it will catch a compromised node as well as its one-hop neighbor node. For a path containing several compromised nodes, it can only identify the first problematic link near the sink.

Fig. 17(a) compares the detection performance between our proposed scheme and the PNM scheme in more details. We can see that our proposed scheme achieves better detection performance after three rounds. Also, the false positive of our proposed scheme is much lower than that of the PNM

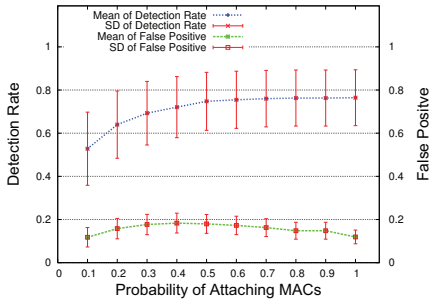
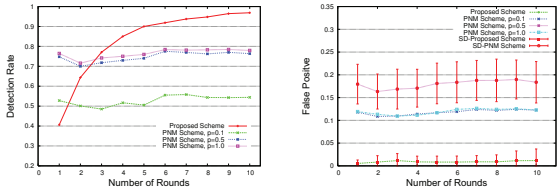


Fig. 16. Performance of the PNM scheme

scheme, as shown by Fig. 17(b). This is because our proposed scheme utilizes the observed wide variety of node behaviors and makes a good heuristic decision on telling which nodes are compromised, while the PNM scheme is only capable to identify the problematic links, and cannot tell which node is the compromised node within a problematic link.



(a) Detection Rate (b) False Positive

Fig. 17. Comparison between collusion and non-collusion

4.3.2 Communication Overhead

We also compare the communication overhead between the PNM scheme and our proposed scheme. In our proposed scheme, only fields R_u and $pad_{u,1}$ are the extra bits, because other fields are necessary even when security issues are not considered (for example, P_u and u are the destination and source of the packet, C_p is the sequence number used by the sink to find out if some packets are lost, etc.). The size of R_u is determined by N_p, which is the maximum number of parents that each sensor node should record at the topology establishment phase. As to be seen later, parameter N_p determines the range that the tree structure can be reshaped; specially, if the total number of nodes in the tree is n, potentially there are N_p^n different tree topologies that can be used to test the behaviors of nodes. Therefore, its value should be reasonably large. It is set to 8 in our simulations. The size of $pad_{u,1}$ is determined by both N_p and h, which is the height of the routing tree. Though the structure of the routing tree changes dynamically from round to round, the level of each node in the tree remains the same. This is because each node only records nodes which are one hop closer to the sink as its candidate parent nodes, and each node dynamically changes its parent node from the recorded candidates in the tree reshaping phase. Therefore, the communication overhead per node is fixed in our proposed scheme.

In the PNM scheme, the extra communication overhead is the marks added for tracing back to the problematic links. In

our simulation, we adopt the RC5 primitives to compute the MAC and the block size is configured to 64 bits. Fig. 18 shows the comparison of extra communication overhead per node. As we can see, the per node communication overhead of the PNM scheme increases as the probability of attaching marks increases. On the other hand, the per node communication overhead of our proposed scheme increases as the number of rounds increases. The proposed scheme outperforms the PNM scheme in terms of communication overhead when the probability of attaching marks is greater than or equal to 0.5. Meanwhile, the detection performance of our proposed scheme also outperforms the PNM scheme, which is shown by Fig. 17.

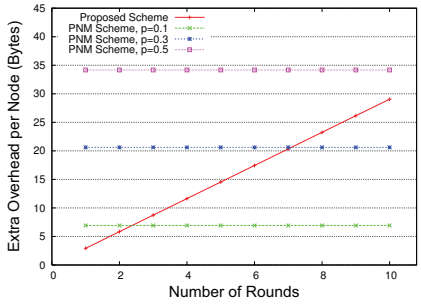


Fig. 18. Comparison of Communication Overhead

4.4 Implementation of Packet Sending and Forwarding

We implemented the proposed packet sending and forwarding scheme on TelosB motes, which are widely used resource-constrained sensor motes [26]. Each TelosB mote has a CPU running at 8MHz, a RAM of 10KB size, and a flash storage of 1MB size. RC5 encryption primitives are used in our implementation. The block size is set to 64 bits. The code running on TelosB consumes 624 bytes of RAM and $15,216$ bytes of ROM. The encryption time on sensor motes depends on the length of encrypted data. In our proposed scheme, the part that needs encryption is $\{R_u, u, C_p \text{ MOD } N_s, D, pad_{u,0}\}_{K_u}$. Its length is decided by several parameters discussed in Sec. 3.1.2. We report the computation overhead in Table 1 by varying the length of data. Other parameters, namely, N_p, N_s, and L_{id} are set to 8, 1024, and 10 bits respectively.

TABLE 1
Computational cost for sensor to forward a packet (ms)

Data Length (Bytes)	12	20	28	36	44	52
Computational Time	120	178	237	296	354	412

From the results shown in Table 1, the computational overhead is quit low, and it is offordable by resource-constrained sensor networks.

5 Related Work

The approaches for detecting packet dropping attacks can be categorized as three classes: multi-path forwarding approach, neighbor monitoring approach and acknowledgement approach. Multi-path forwarding [4], [5] is a widely adopted countermeasure to mitigate packet droppers, which is based on delivering redundant packets along multiple paths. Another approach is to exploit the monitoring mechanism [10], [13], [14], [16], [17], [18], [19], [27]. The *watchdog* method was originally proposed to mitigate routing misbehavior in mobile ad hoc networks [10]. It is then adopted to identify packet droppers in wireless sensor network [13], [27], [28]. When the watchdog mechanism is deployed, each node monitors its neighborhood promiscuously to collect the firsthand information on its neighbor nodes. A variety of reputation systems have been designed by exchanging each node's first-hand observations, which are further used to quantify node's reputation [16], [17], [18], [19]. Based on the monitoring mechanism, the intrusion detection systems are proposed in [15], [29]. However, the watchdog method requires nodes to buffer the packets and operate in the promiscuous mode, the storage overhead and energy consumption may not be affordable for sensor nodes. In addition, this mechanism relies on the bidirectional communication links, it may not be effective when directional antennas are used [30]. Particularly, this approach cannot be applied when a node does not know the expected output of its next hop since the node has no way to find a match for buffered packets and overheard packets. Note that, this scenario is not rare, for example, the packets may be processed, and then encrypted by the next hop node in many applications that security is required. Since the watchdog is a critical component of reputation systems, the limitations of the watchdog mechanism can also limit the reputation system. Besides, a reputation system itself may become the attacking target. It may either be vulnerable to bad mouthing attack or false praise attack [30]. The third approach to deal with packet dropping attack is the multi-hop acknowledgment technique [31], [32], [33]. By obtaining responses from intermediate nodes, alarms and detection of selective forwarding attacks can be conducted. To deal with packet modifiers, most of existing countermeasures [6], [7], [8], [9] are to filter modified messages within a certain number of hops so that energy will not be wasted to transmit modified messages.

The effectiveness to detect malicious packet droppers and modifiers is limited without identifying them and excluding them from the network. Researchers hence have proposed schemes to localize and identify packet droppers, one approach is the acknowledgement based scheme [24], [25], [34] for identifying the problematic communication links. It can deterministically localize links of malicious nodes if every node reports ACK using onion report. However, this incurs large communication and storage overhead for sensor networks. The probabilistic ACK approaches are then proposed in [24], [25], which seek tradeoffs among detection rate, communication overhead and storage overhead. However, these approaches assume the packet sources are trustable, which may not be valid in sensor networks. As in sensor networks, base station typically is the only one we can trust. Furthermore, these schemes require to set up pairwise keys among regular sensor nodes so as to verify the authenticity of ACK packets, which may cause considerable overhead for key management in

sensor networks. Ye *et al.* [20] proposed a scheme called PNM for identifying packet modifiers probabilistically. However, the PNM scheme cannot be used together with the false packet filtering schemes [6], [7], [8], [9], 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.

6 Conclusion

We propose a simple yet effective scheme to identify misbehaving forwarders that drop or modify packets. Each packet is encrypted and padded so as to hide the source of the packet. The packet mark, a small number of extra bits, is added in each packet such that the sink can recover the source of the packet and then figure out the dropping ratio associated with every sensor node. The routing tree structure dynamically changes in each round so that behaviors of sensor nodes can be observed in a large variety of scenarios. Finally, most of the bad nodes can be identified by our heuristic ranking algorithms with small false positive. Extensive analysis, simulations and implementation have been conducted and verified the effectiveness of the proposed scheme.

Acknowledgments

This work is partially supported by NSF under grants CNS-0716744, CNS-0627354, CNS-0834593 and CNS-0831874, and by ONR under grant N000140910748.

References

[1] H. Chan and A. Perrig, "Security and privacy in sensor networks," *Computer*, vol. 36, no. 10, 2003.

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

[3] V. Bhuse, A. Gupta, and L. Lilien, "Dpdsn: Detection of packet-dropping attacks for wireless sensor networks," in *the Fourth Trusted Internet Workshop*, 2005.

[4] M. Kefayati, H. R. Rabiee, S. G. Miremadi, and A. Khonsari, "Misbehavior resilient multi-path data transmission in mobile ad-hoc networks," in *ACM SASN*, 2006.

[5] R. Mavropodi, P. Kotzanikolaou, and C. Douligeris, "Secmr - a secure multipath routing protocol for ad hoc networks," *Ad Hoc Networks*, vol. 5, no. 1, 2007.

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

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

[8] H. Yang, F. Ye, Y. Yuan, S. Lu, and W. Arbaugh, "Toward Resilient Security in Wireless Sensor Networks," in *ACM MobiHoc*, 2005.

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

[10] S. Marti, T. Giuli, K. Lai, and M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks," in *ACM MobiCom*, 2000.

[11] M. Just, E. Kranakis, and T. Wan, "Resisting malicious packet dropping in wireless ad hoc networks," in *ADHOCNOW*, 2003, vol. 2856.

[12] R. Roman, J. Zhou, and J. Lopez, "Applying intrusion detection systems to wireless sensor networks," in *IEEE CCNC*, 2006.

[13] S. Lee and Y. Choi, "A resilient packet-forwarding scheme against maliciously packet-dropping nodes in sensor networks," in *ACM SASN*, 2006.

[14] I. Khalil and S. Bagchi, "Mispar: mitigating stealthy packet dropping in locally-monitored multi-hop wireless ad hoc networks," in *SecureComm*, 2008.

[15] I. Krontiris, T. Giannetsos, and T. Dimitriou, "Lidea: a distributed lightweight intrusion detection architecture for sensor networks," in *SecureComm*, 2008.

[16] S. Ganeriwal, L. K. Balzano, and M. B. Srivastava, "Reputation-based framework for high integrity sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 4, no. 3, 2008.

[17] W. Li, A. Joshi, and T. Finin, "Coping with node misbehaviors in ad hoc networks: A multi-dimensional trust management approach," in *IEEE Mobile Data Management*, 2010.

[18] P. Michiardi and R. Molva, "Core: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks," in *CMS*, 2002.

[19] S. Buchegger and J. Le Boudec, "Performance analysis of the confidant protocol," in *ACM MobiHoc*, 2002.

[20] F. Ye, H. Yang, and Z. Liu, "Catching Moles in Sensor Networks," in *IEEE ICDCS*, 2007.

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

[22] K. Sun, P. Ning, C. Wang, A. Liu, and Y. Zhou, "Tinysersync: Secure and resilient time synchronization in wireless sensor networks," in *ACM CCS*, 2006.

[23] H. Song, S. Zhu, and G. Cao, "Attack-resilient time synchronization for wireless sensor networks," *Ad Hoc Networks*, vol. 5, no. 1, 2007.

[24] B. Xiao, B. Yu, and C. Gao, "Chemas: Identify suspect nodes in selective forwarding attacks," *Journal of Parallel and Distributed Computing*, vol. 67, no. 11, 2007.

[25] X. Zhang, A. Jain, and A. Perrig, "Packet-dropping adversary identification for data plane security," in *ACM CONEXT*, 2008.

[26] Crossbow, "Wireless sensor networks," *http://www.xbow.com/Products/Wireless_Sensor_Networks.htm*.

[27] T. H. Hai and E. N. Huh, "Detecting selective forwarding attacks in wireless sensor networks using two-hops neighbor knowledge," in *IEEE NCA*, 2008.

[28] F. Liu, X. Cheng, and D. Chen, "Insider Attacker Detection in Wireless Sensor Networks," in *IEEE INFOCOM*, 2007.

[29] K. Ioannis, T. Dimitriou, and F. C. Freiling, "Towards intrusion detection in wireless sensor networks," in *EuropeanWireless Conference*, 2007.

[30] A. Srinivasan, J. Teitelbaum, H. Liang, J. Wu, and M. Cardei, "Reputation and trust-based systems for ad hoc and sensor networks," in *Algorithms and Protocols for Wireless Ad Hoc and Sensor Networks,*. 2008, Wiley & Sons.

[31] J. M. Mccune, E. Shi, A. Perrig, and M. K. Reiter, "Detection of denial-of-message attacks on sensor network broadcasts," in *IEEE S&P*, 2005.

[32] B. Yu and B. Xiao, "Detecting selective forwarding attacks in wireless sensor networks," in *IPDPS*, 2006.

[33] K. Liu, J. Deng, P. K. Varshney, and K. Balakrishnan, "An acknowledgment-based approach for the detection of routing misbehavior in manets," *Mobile Computing, IEEE Transactions on*, vol. 6, no. 5, 2007.

[34] B. Barak, S. Goldberg, and D. Xiao, "Protocols and lower bounds for failure localization in the internet," in *Eurocrypt*, 2008.