On the Intruder Detection for Sinkhole Attack in Wireless Sensor Networks

Edith C. H. Ngai,[1] Jiangchuan Liu,[2] and Michael R. Lyu[1]

[1]Department of Computer Science and Engineering, The Chinese University of Hong Kong
[2]School of Computer Science, Simon Fraser University, British Columbia, Canada

Abstract— **In a wireless sensor network, multiple nodes would send sensor readings to a base station for further processing. It is well-known that such a many-to-one communication is highly vulnerable to the *sinkhole attack*, where an intruder attracts surrounding nodes with unfaithful routing information, and then performs selective forwarding or alters the data passing through it. A sinkhole attack forms a serious threat to sensor networks, particularly considering that such networks are often deployed in open areas and of weak computation and battery power.**

In this paper, we present a novel algorithm for detecting the intruder in a sinkhole attack. The algorithm first finds a list of suspected nodes, and then effectively identifies the intruder in the list through a network flow graph. The algorithm is also robust to deal with cooperative malicious nodes that attempt to hide the real intruder. We have evaluated the performance of the proposed algorithm through both numerical analysis and simulations, which confirmed the effectiveness and accuracy of the algorithm. Our results also suggest that its communication and computation overheads are reasonably low for wireless sensor networks.

I. Introduction

Wireless sensor networks become increasingly popular to solve such challenging real-world problems as industrial sensing and environmental monitoring. A sensor network generally consists of a set of sensor nodes, which continuously monitor their surroundings and forward the sensing data to a sink node, or base station. It is well-known that such a many-to-one communication is highly vulnerable to the *sinkhole attack*, where an intruder attracts surrounding nodes with unfaithful routing information, and then alters the data passing through it or performs selective forwarding.

A sinkhole attack prevents the base station from obtaining complete and correct sensing data, and thus forms a serious threat to higher-layer applications. It is particularly severe for wireless sensor networks given the vulnerability of wireless links, and that the sensors are often deployed in open areas and of weak computation and battery power. Although some secure or geographic based routing protocols resist to the sinkhole attacks in certain level [1], many current routing protocols in sensor networks are susceptible to the sinkhole attack [2].

The work described in this paper was substantially supported by grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK4205/04E).
J. Liu's work was supported in part by a Canadian NSERC Discovery Grant 288325, an NSERC Research Tools and Instruments Grant, a Canada Foundation for Innovation (CFI) New Opportunities Grant, and an SFU President's Research Grant.

In this paper, we propose a novel light-weighted algorithm for detecting sinkhole attacks and identifying the intruder in an attack. We focus on a general many-to-one communication model, where the routes are established based on the reception of route advertisements. Our solution explores the asymmetric property between the sensor nodes and the base station, and makes effective use of the relatively-high computation and communication power in the base station [2, 3, 4]. It consists of two steps: First, a secure and low-overhead algorithm for the base station to collect the network flow information with a distributed fashion in the attack area; and second, an efficient identification algorithm that analyzes the collected network flow information and locate the intruder. We also consider the scenario that a set of colluding nodes cheat the base station about the location of the intruder. Specifically, we examine multiple suspicious nodes and conclude the intruder based on majority votes. We show that such a conclusion is correct if less than half of the collected information comes from malicious nodes.

The performance of the proposed algorithm is evaluated through both numerical analysis and simulations, which confirmed the effectiveness and accuracy of the algorithm. Our results also suggest that its communication and computation overheads are reasonably low for wireless sensor networks.

The remainder of this paper is organized as follows. Section II presents the related work. In Section III, we formally describe the sinkhole attack in wireless sensor networks, and state the problem to be solved. In Section IV, we present our 2-step detection algorithm, i.e., collecting network flow information and identifying the intruder. In Section V, we provide the enhancements to the algorithm for handling multiple malicious nodes and give the numerical analysis. The performance of the proposed algorithm is evaluated in Sections VI through simulations. Finally, Section VII concludes this paper and offers some future research directions.

II. Related Work

Intrusion detection has been an active research topic for the Internet extensively [5]. Recently, many detection algorithms have been proposed for wireless ad hoc networks as well. Most of them assume uniform nodes and symmetric communications [6]. On the contrary, the sensor network we are considering has an asymmetric many-to-one communication pattern, and the power of the sensor nodes is rather weak.

1-4244-0355-3/06/$20.00 (c) 2006 IEEE

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2006 proceedings.

For sensor networks, some existing secure or geographical routing protocols are resistant to sinkhole attack in certain level. An example is a geographic protocol [1], which performs routing by the localized information and interactions only, without an initiation from the base station. However, many of the existing routing protocols, in particular, those based on route advertisement, are vulnerable to sinkhole attacks. To the best of our knowledge, we are not aware of any algorithm that is specifically designed for sinkhole detection among them.

Our work is also motivated by the following studies, though they have focused on different applications. Specifically, Wood et al. [7] proposes a mechanism for detecting and mapping jammed regions. They describe a mapping protocol for nodes that surround a jammer which allows network applications to reason about the region as an entity, rather than as a collection of broken links and congested nodes. Ding et al. [8] propose an algorithm for the identification of faulty sensors and detection of the reach of events in sensor networks with faulty sensors. Staddon et al. [9] demonstrate that the topology of the network can be efficiently conveyed to the base station allowing for the quick tracing of the identities of the failed nodes with moderate communication overhead. Ye et al. [10] present a Statistical En-route Filtering (SEF) mechanism that can detect and drop such false reports; SEF applies multiple keyed message authentication codes, probabilistic verification, and data filtering to determine the truthfulness of each report. Perrig et al. [11] propose a packet leash mechanism for detecting and thus defending against wormhole attacks. A leash can be some temporal or geographical information that is added to a packet to restrict the packet's maximum allowed transmission distance.

III. Problem Statement

We consider a sensor network that consists of a base station (BS) and a collection of geographically distributed sensor nodes, each denoted by a unique identifier ID_v. The sensor nodes continuously collect and send the sensed application data to the base station by forwarding packets hop-by-hop.

As mentioned earlier, this commonly used many-to-one communication pattern is vulnerable to sinkhole attacks. In a sinkhole attack, an intruder usually attracts network traffic by advertising itself as having the shortest path to the base station. For example, as shown in Figure 1a, an intruder using a wireless-enabled laptop will have much higher computation and communication power than a normal sensor node, and it could have a high-quality single-hop link to the base station (BS). It can then advertise imitated routing messages about the high quality route, thus spoofing the surrounding nodes to create a sinkhole (SH). A sinkhole can also be performed using a wormhole [12], which creates a metaphorical sinkhole with the intruder being at the center. An example is shown in Figure 1b, where an intruder creates a sinkhole by tunneling messages received in one part of the network and replays them in a different part using a wormhole.

We assume the sensor nodes are either *good* or *malicious*. The center of a sinkhole attack is a malicious node compromised by the intruder. Note that, even if there is only one compromised node providing a high quality route to the base station, it can affect many surrounding sensors. Furthermore, this intruder may also cooperate with some other malicious nodes in the network to interfere detection algorithms. In an extreme case, all the malicious nodes are colluding with the intruder. They may collaboratively cheat the base station by claiming a good node as the intruder (the victim, SH'), and thus hide the real one.

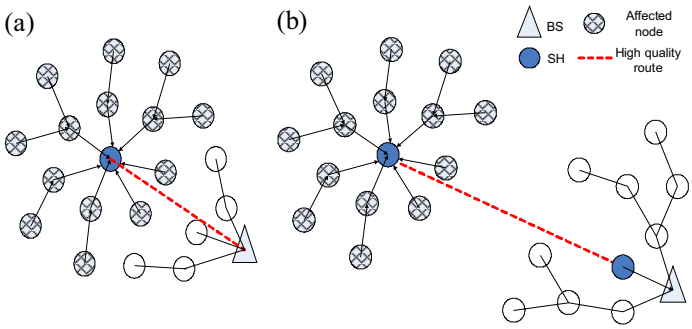

Fig. 1. Two examples of sinkhole attack in wireless sensor networks. (a) Using an artificial high quality route; (b) Using a wormhole.

The focus of our work is to effectively identify the real intruder (SH) in the sinkhole attack in presence of colluding nodes. We assume that the base station is physically protected or has tamper-robust hardware [3]; hence, it acts as a central trusted authority in our algorithm design. The base station also has a rough understanding on the location of nodes, which could be available after the node deployment stage or can be obtained by various localization mechanisms [13].

For ease of exposition, in Table I, we list the major notations used throughout this paper.

TABLE I
List of Notation

BS	Base station
SH	Real intruder in the sinkhole attack
SH'	False intruder (victim) in the sinkhole attack
ID_v	Identity of sensor node v
p	Probability of malicious in sensor nodes
d	Packet drop rate
k	No. neighbors which a message will be forwarded to
h_{max}	Hops from the farthest node to BS
h_{rc}	Hops from BS where root correction takes place
l	Levels from BS in the tree of network flow
t^l	No. of nodes at level l
N	Total number of nodes in the attack area
M	No. of malicious nodes in the attack area
r	No. of correct network flow information collected
m	No. of incorrect network flow information collected
s	No. of missing network flow information
n	Total no. of network flow information collected

IV. Sinkhole Attack Detection

In this section, we describe how to detect a sinkhole attack, and then efficiently identify the intruder. We first focus on one malicious node (the intruder); an enhancement dealing with multiple malicious nodes will be presented in the next section.

A. Estimating the Attacked Area

In many sensor network applications, sensor nodes are responsible for collecting local data and sending them to the *BS*. The most common kind of violations in a sinkhole attack is selective forwarding. By observing consistent data missing from an area, the *BS* may suspect there is an attack with selective forwarding. For illustration, consider a monitoring application in which sensor nodes submit sensing data to the *BS* periodically. The *BS* can detect the data inconsistency using the following statistical method. Let $X_1, \ldots, X_n$ be the sensing data collected in a sliding window, and $\bar{X}$ be their mean. Define $f(X_j)$ as

$$f(X_j) = \sqrt{\frac{(X_j - \bar{X})^2}{\bar{X}}}$$

Then a simple measure for identifying a suspected node is if $f(X_j)$ is greater than a certain threshold, because the data from this node is inconsistent with others in the same area. More advanced techniques can be found in [14, 15].

After identifying a list of the suspected nodes, the *BS* can estimate where the sinkhole locates. Specifically, it can circle a potential *attacked area*, which contains all the suspected nodes. The radius of the circle can be selected for just covering all the suspected nodes. An example is shown in Figure 2, where the shaded nodes are found to have missing or inconsistent data. Note that all the nodes in the circle could be attracted by the sinkhole, and hence, they are referred to as *affected nodes*.

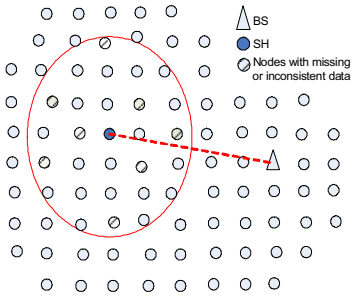

Fig. 2. Estimate the attacked area

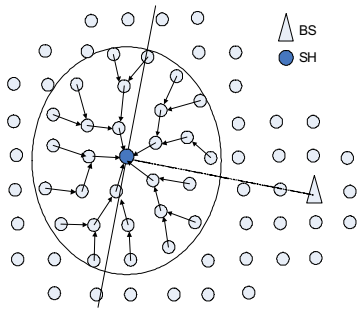

Fig. 3. Network flow in the attacked area

B. Identifying the Intruder

Since the attacked area may contain many nodes, and the sinkhole is not necessarily the center of the area, it is better to further locate the exact intruder and isolate it from the network. This can be achieved through analyzing the routing pattern in the affected area.

We first demonstrate how to collect the network flow information. The *BS* sends a request message to the network. The request message contains the *ID*s of all the affected nodes, and is flooded hop by hop. To prevent affected nodes to exploit replay attacks on network flow information request messages, the *BS* can include a timestamp *TS* and sign the message with its private key K_{BS}. The format of the request message is $<TS, ID_1, \ldots, ID_n>_{K_{BS}}$. For each node v receiving the first request, if its *ID* is there, it should reply the *BS* a message $<ID_v, ID_{next\text{-}hop}, cost>$, which includes its own *ID*, the *ID* of the next-hop node, and the cost, e.g., hop-count, data rate, etc. The *ID* of the next-hop node and the cost are stored by individual nodes according to their routing protocol. Note that the next-hop and the cost could already be affected by the attack; hence, the reply message should be sent along the reverse path in the flooding, which corresponds to the original route with no intruder.

At the *BS*, each piece of network flow information can be represented by a directed edge, i.e. $a \xrightarrow{ct} b$, where a denotes an affected node, b denotes the next hop of a, and ct denotes the cost from a to the *BS*. The next hop information is important for observing the routing pattern. The *BS* can realize the routing pattern by constructing a tree using the next hop information collected. An area invaded by a sinkhole attack processes special routing pattern where all network traffic flows toward the same destination, which is compromised by the intruder *SH*. It is also the root in the tree of network flow information as shown in Figure 3. Apart from the essential next hop information, additional routing information such as hop count can facilitate the intruder detection by checking any erroneous or inconsistent flow information. It is especially helpful when the attacked area is in presence of multiple malicious nodes, as will be discussed in the next section.

Due to the missing information, some branches of the network flow information tree constructed may be broken. The *BS* may obtain more than one tree of network flow information. Algorithm 1 shows the construction of the trees in the attacked area. We can then calculate the number of nodes in different trees by a depth-first search, and the intruder should be the root of the biggest tree, which attracts most network traffic.

Algorithm 1 Identify multiple roots

```
R = φ                    /* R: set of roots */
for each v ∈ S    /* S: set of nodes in the attacked area */
    if v has no incoming edge
        R = R ∪ findR(v)
end for

findR(node u)
        R' = φ
        if u is not yet visited
            mark u is visited
        else
            return φ
        if u has no outgoing edge
            return {u}
        for each e(u,v)
            R' = R' ∪ findR(v)
        end for
        return R'
end findR
```

V. Enhancements against Multiple Malicious Nodes

A. Enhancements on Network Flow Information Collection

As mentioned before, there could be multiple malicious nodes that prevent the *BS* from obtaining correct and complete flow information for intruder detection. They may cooperate with the intruder to perform the following misbehaviors:

1. Modify the packets passing through
2. Forward the packets selectively
3. Provide wrong network flow information of itself

We now show effective enhancements that address these issues through encryption and path redundancy.

1) Key Establishment

Applying encryption can avoid tampering of the packets during transmission. We assume every node v has an individual key K_v that is shared with the *BS* only. This key is loaded to the node through a pre-distribution protocol [16]. In order to save storage space at the *BS*, the pairwise key K_v is derived by a master key K_{BS}, using a pseudo-random function F and the unique node ID_v: $K_v = F_{Kv}(ID_v)$.

Given its key, a sensor node encrypts its packet when it replies network flow information to the *BS*. Existing studies have shown that a symmetric encryption can be efficiently implemented in various small sensing devices [17, 18]. The node can also use one-way hash chains to provide authentication [18]. For example, a sender selects a random value K_q as the last key in the key chain and repeatedly performs a pseudo random function F to computer all other keys: $K_q = F(K_{q+1})$, where K_q is the secret key assigned to the *q-th* time interval.

2) Path Redundancy

Since malicious nodes may drop the reply messages, sensors can forward network flow information to the *BS* through multiple redundant paths. Specifically, they can forward reply messages to k neighbors, where $k \geq 1$.

Let k be the number of neighbors to which a message is forwarded towards the *BS*, p be the probability of malicious in a sensor node, h_{max} be the number of hops from the farthest node to the *BS*, and level l has t^l nodes, where $1 \leq l \leq h_{\max}$. We have

Pr[a response message from a node at level l can reach *BS*] $= (1-p)(1-p^k)^{l-1}$

And the number of responses reaching the *BS* is thus

$$n = \sum_{l=1}^{h_{max}} (1-p)(1-p^k)^{l-1} t^l$$

As an example, for p=0.1, k=2, h_{max}=3, t=4, we have n= 73.31 out of 84*(1-0.1)=75.6 responses being sent out by the honest nodes, which is reasonably good.

B. Dealing with Multiple Malicious Nodes

Multiple malicious nodes may refuse to send reply message or drop some of the reply packets (Figure 4a). Besides, they may provide incorrect flow information, like next hop or hop count, cooperatively to manipulate intruder detection (Figure 4b). Their objective is to hide the real intruder *SH* and blame on a victim node *SH'*. An example is shown in Figure 4b, where two colluding nodes A and C provide an outgoing edge to a victim node *SH'*. To deal with this problem, the *BS* detects the inconsistency among the hop count information. For instance, nodes D, E, and F have same number of hop counts in their incoming and outgoing edges, which is suspicious. Moreover, the incoming edges of *SH'* have different number of hop counts. In our algorithm, we calculate the difference between the hop count provided by a node and the number of edges from the node to the current root. By spotting the inconsistency of the hop counts, we could identify *SH* and other suspicious nodes.

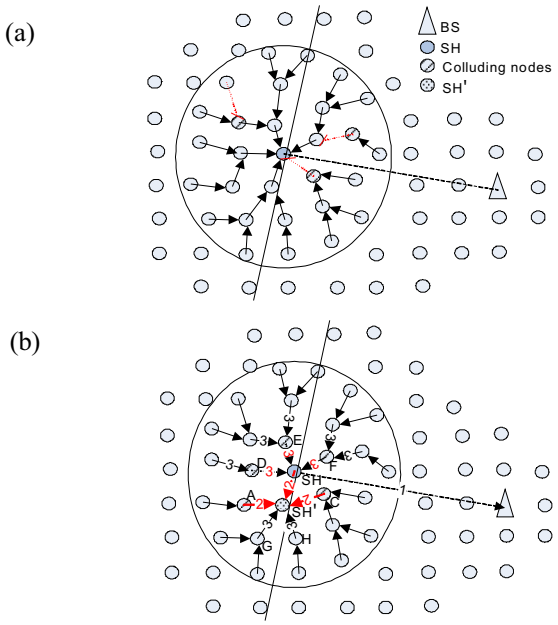

Fig. 4. Attack area with colluding nodes (a) Missing information; (b) Misleading network flow

To this end, we maintain an array *Count*[], where entry *Count*[*i*] stores the total number of nodes having hop count difference *i*. Note that index *i* can be negative, which indicates that the hop count provided by a node is smaller than its actual distance from the current root. Intuitively, if *Count*[0] is not the dominated one in the array, it means the current root is unlikely the real intruder. By analyzing the array *Count*, we may estimate the hop counts from *SH'* to *SH*. For example, if most non-zero entries of *Count* fall in the index range [-2, 2], we may suspect *SH* is two hops away from *SH'* (the current root). Based on this, the *BS* can make root correction and re-calculate the array *Count* among the nodes within two hops from *SH'* (Algorithm 2). Finally, it concludes the intruder based on the most consistent result. Figure 5 shows an example of the root correction algorithm. Node *SH'* is the original root of the network flow tree (Figure 5a). Its array *Count* is as follow:

i	-2	**-1**	**0**	1	2
Count[i]	0	**14**	**8**	6	0

It shows that only 8 nodes agree that *SH'* is the intruder. However, 14 nodes do not agree with that. Instead, they believe *SH* should be one hop closer to *BS* than node *SH'*. Since they are the majority, our correction algorithm has to run again and look for a new root. Eventually, node *SH* becomes the new root after the correction algorithm (Figure 5b), and the corresponding new array *Count* is as follow:

i	-2	-1	**0**	1	2
Count[i]	0	1	**21**	6	0

It shows that 21 nodes provide consistent information about the current root *SH*. Since the value of *Count*[0] is the majority, *SH* is concluded as the intruder.

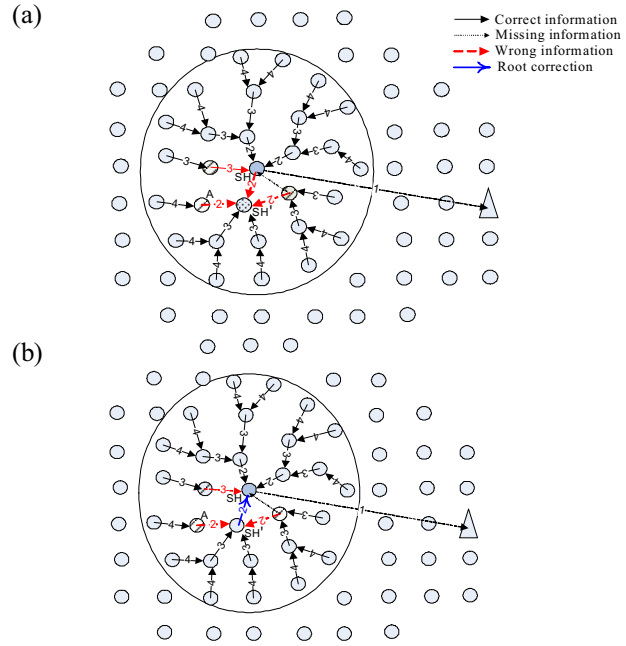

Fig. 5. Example of intruder identification with multiple malicious nodes

Algorithm 2 Find the real intruder with root corrections

```
for each root r
    initialize a new Array count
    initialize a new Path correctPath
    checkRootByCount(r, count, 1)
    S = {x>0 | forall y>0, count[x]+count[-x]>count[y]+count[-y]}
    x = min (S)
    correctRoot(r, r, x, 0, correctPath , count[0])
    apply correctPath on Network G
end for

checkRootByCount (Node r, Array count, int depth)
    depth = depth +1
    for each precedent Node c of r
        increase count[hop_count(c) – depth] by 1
        checkRootByCount (c, count, depth)
    end for
end checkRootByCount

correctRoot(Node r, Path p, int totalLevel, int currentLevel, Path
correctPath, int bestCount)
    if (currentLevel >= totalLevel)
        return
    end if
    currentLevel= currentLevel+1
    for each precedent node c of r
        initialize a new Array count
        reverse edge (c,r)
        checkRootByCount (c, count, 1)
            if (count[0]> bestCount)
                    correctPath = p->c
                    bestCount = count[0]
            end if
        correctRoot(c, p->c, totalLevel, currentLevel, correctPath ,
        bestCount)
        reverse edge(c,r)
    end for
end correctRoot
```

Our approach supports the detection of multiple sinkholes in the network. Intruder identification algorithm can be applied to each attack area separately. Our approach also works for highly clustered or more hierarchical sensor networks, provided that the network flow information of the attack area can reach the base station.

It can be seen that the time complexity for calculating the array count is $O(N)$, and that for correcting the roots is $\sum_{l=1}^{h_{rc}} t^l \cdot N = O(t^{h_{rc}} \cdot N)$, where h_{rc} is the average number of hops where a root correction will take place. The total time is thus $O(N) + O(N) + O(t^{h_{rc}} \cdot N) = O(t^{h_{rc}} \cdot N)$, which is relatively low in terms of energy consumption, as will be examined in the next section.

C. Analysis

We now give a simple analysis about the effectiveness of the above algorithm. We assume that N is the number of nodes in the attacked area, $N = n + s = (m + r) + s$.

Property: For any m, our sinkhole detection algorithm works if there are more than $2m$ sensors provide network flow information successfully to *BS* and at most m are malicious among them.

Proof: Since there are more than $2m$ sensors successfully providing their network flow information, we have $n > 2m$, and consequently $r > m$.

In this case, there are more honest sensors than malicious sensors successfully providing their network flow information to the *BS*. In the worst case, all the m malicious sensors are colluding and suggesting *SH'* as the intruder. Yet, the r good sensors will suggest *SH*, the real intruder, and hence our algorithm is still able to make a correct conclusion, even in this worst case.

Our algorithm might not work if $n \leq 2m$ and more than m malicious nodes are colluding. Nevertheless, in most sensor networks, a majority of sensors should be in normal condition and such situation thus will seldom happen.

TABLE II
ENVIRONMENT SETTING OF THE EXPERIMENTS

No. of nodes in network	400
Size of network	200m x 200m
Transmission range	10m
Location of *BS*	(100,100)
Location of sinkhole	(50, 50)
Percentage of colluding codes (m)	0 – 50%
Message drop rate (d)	0 – 80%
No. of neighbors which a message is forwarded to (k)	1 – 2
Packet size	100bytes
Max. number of reply messages per packet	5

VI. PERFORMANCE EVALUATION

We further evaluate the performance of our sinkhole detection algorithm through simulations. We are interested in its accuracy on intruder identification, communication overhead, and energy consumption. We implement our own experiments with static sensors uniformly placed in the network area. Table II shows the environment settings of our implementation [10, 7]. The sinkhole attack is simulated by an intruder *SH* that attracts network traffic from its surrounding nodes. There are 50 nodes being affected by the sinkhole and m of them is colluding with the intruder. Messages may be randomly dropped at rate d during network flow information collection. To mitigate this loss, a sensor node forwards its reply message to k neighbors. It stores up to five messages before packing them into one packet and forwarding to the next hop.

A. *Accuracy of Intruder Identification*

This experiment investigates the accuracy on intruder identification for the sinkhole attacks. The success rate represents the percentage that our algorithm can correctly identify the *SH*, the false-positive rate represents the percentage that our algorithm identifies *SH'* falsely, and the false-negative rate represents the percentage that our algorithm is not able to identify any sinkhole but it exists.

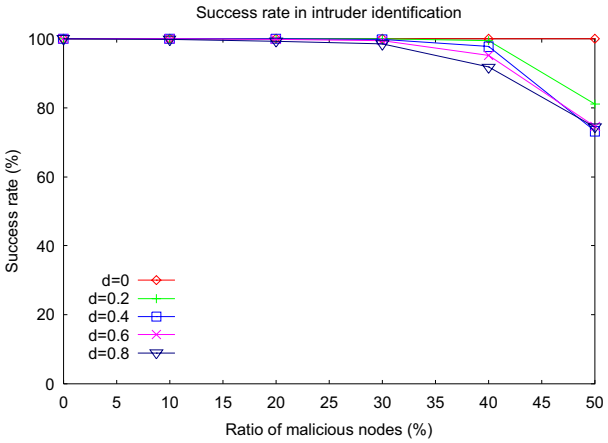

Fig. 6. Success rate in intruder identification

Figure 6 shows the success rate of intruder identification. The experiment result shows that our identification algorithm works well when m is less than 50%. All intruders can be identified accurately when d=0. However, the success rate drops when the dropping rate increases. It is because some network flow information is missing, so the intruder may be falsely identified. The success rate also decreases when the malicious rate increases. This is due to the increased amount of misleading network flow information from the colluding nodes.

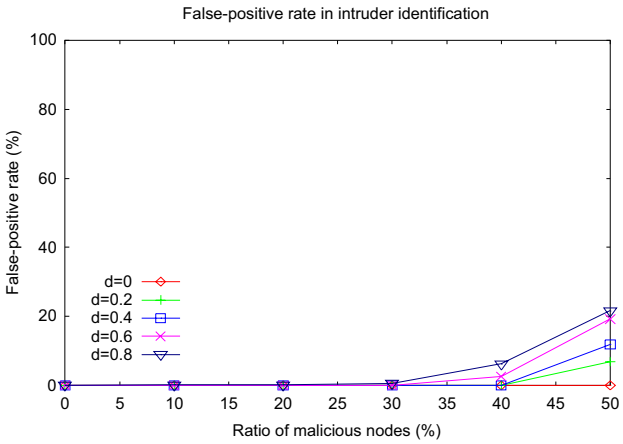

Fig. 7. False-positive rate in intruder identification

Figure 7 and Figure 8 show the false-positive rate and false-negative rate in intruder identification respectively. The simulation results indicate that the error rates are quite low. There is no false-positive and false-negative errors when $d = 0$. The error rates increase slightly with increasing the dropping rate and the malicious rate. When the number of colluding nodes increases, there is more incorrect network flow information provided to the *BS*. If many correct messages are dropped, the remaining wrong information can mislead the *BS*. The *BS* may incorrectly conclude an intruder and lead to a false-positive error. Similarly, the *BS* may receive inadequate number of messages to identify the intruder and bring a false-negative error.

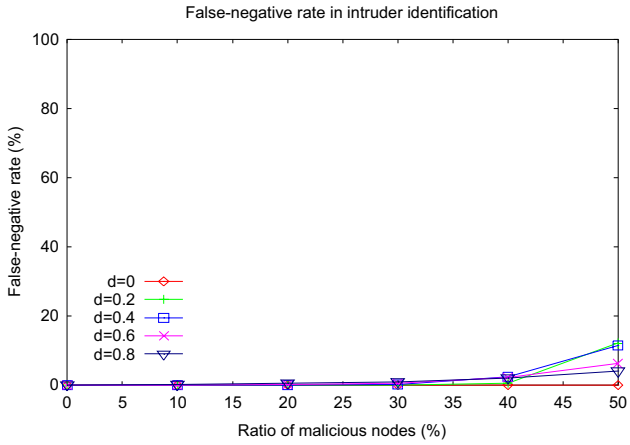

Fig. 8. False-negative rate in intruder identification

B. *Communication Cost*

In this experiment, we evaluated the communication overhead of algorithm, in which the overhead for collecting network flow information is dominating. Figure 9 shows the number of packets send or receive by nodes with various hops to the *BS*. Nodes closer to the *BS* are shown to have more overheads. The variable k represents the number of neighbors to which a message will be forwarded. As an example, when k is *2*, for a node in the attacked area, the packets sent are around a double of the packets received. Note that the numbers of packets sent and received are the same for hop count less than *4*. This is because most malicious nodes are located in the attacked area, and the nodes out of the area only forward packets to one neighbor to relieve the bottleneck around the *BS*. The larger the value of k is, the less the network flow information will be lost for intruder identification by means of the redundant paths. Thus, there is a tradeoff between the communication overhead and the accuracy on intruder identification. Our experience shows that, when k=2, the overhead is reasonably low while a high security level can be expected.

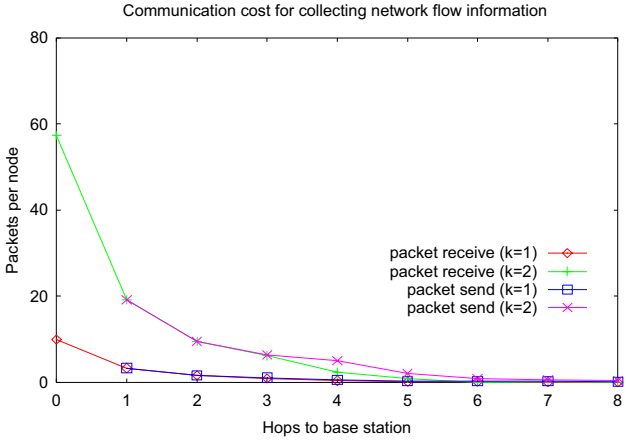

Fig. 9. Communication cost for collecting network flow information

C. Energy Consumption

Finally, we study the energy consumption in a sensor node for intruder identification. It is related to both the communication overhead for message passing and the computation overhead for encrypting and signing messages. Table III shows typical energy consumptions for a sensor node, which are adapted from [19, 4]. We used these parameters to calculate the energy consumption of individual sensors for receiving, sending, and encryption the messages related to intruder identification. Note that computation overhead of encrypting and signing the messages is less than 5% of the total energy consumption, which is consistent with the observation in [4].

TABLE III
PARAMETERS OF ENERGY CONSUMPTION

Communication circuit power	5×10^{-8} J/bit
Communication antenna power	1×10^{-10} J/bit/m^2
Encryption and MAC computation	3×10^{-9} J/bit

Figure 10 shows the average energy consumption for our algorithm at each single node as a function of its hop counts to the *BS*. It can be seen that the nodes located closer to the *BS* experience higher energy consumption. This is because they become the bottleneck between the *BS* and the attack area. They consume more energy for sending and receiving a greater amount of messages than other nodes. The number of messages also increases with k, because a node may forward its packets to more than one neighbor. Applying redundant paths require more energy consumption, but it provides higher accuracy of intruder identification by diminishing the impact of message dropping. Nonetheless, the energy consumption for our intruder detection algorithm is quite lightweighted. For example, consider a sensor node with two 3V off-the-shelf, 1.2 Amp-Hour batteries [20, 21]. The total energy available at this node is $Et = 2 * 3 * 1.2 = 7.2$ in *V·A·Hour*. Clearly, it needs to spend only a very small portion of the available energy for intruder identification throughout its lifetime.

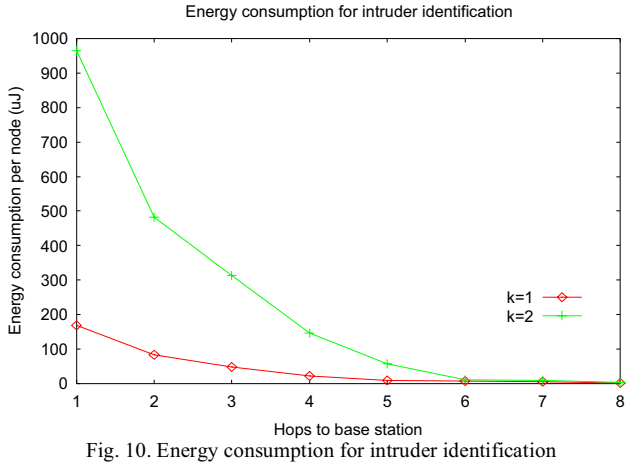

Fig. 10. Energy consumption for intruder identification

VII. CONCLUSION AND FUTURE WORK

In this paper, we have presented an effective method for identifying sinkhole attack in a wireless sensor network. The algorithm consists of two steps. It first locates a list of suspected nodes by checking data consistency, and then identifies the intruder in the list through analyzing the network flow information. We have also presented a series enhancements to deal with cooperative malicious nodes that attempt to hide the real intruder.

The performance of the proposed algorithm has been examined through both numerical analysis and simulations. The results have demonstrated the effectiveness and accuracy of the algorithm. They also suggest that its communication and computation overheads are reasonably low for wireless sensor networks. There could be many future directions toward enhancing this work. In particular, we are interested in more effective statistical algorithms for identifying data inconsistency, and thus correctly locating suspected nodes in sinkhole attacks.

REFERENCES

[1] Brad Karp and H. T. Kung, "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks," in *Proc. of the 6th ACM MobiCom,* Aug 2000, pp. 243-254.

[2] Karlof and D. Wagner, "Secure Routing in Sensor Networks: Attacks and Countermeasures," in *Proc. of the 1st IEEE Workshop on Sensor Network Protocols and Applications*, May 2003, pp.1-15.

[3] E. Shi and A. Perrig, "Designing Secure Sensor Networks," *IEEE Wireless Communications*, vol. 11, no. 6, Dec 2004, pp. 38-43.

[4] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "SPINS: Security Protocols for Sensor Networks," in *Proc. of the 7th ACM MobiCom,* Jul 2001, pp. 189-199.

[5] Dorothy E. Denning, "An Intrusion Detection Model," *in Proc. of the IEEE Symposium on Security and Privacy*, 1986, pp. 118-131.

[6] Y. Zhang and W. Lee, "Intrusion Detection in Wireless Ad-Hoc Networks," in *Proc. of the 6th ACM MobiCom,* Aug 2000, pp. 275-283.

[7] A. D. Wood, J. A. Standovic, and S. H. Son, "JAM: A Jammed-Area Mapping Service for Sensor Networks," in *Proc. of the 24th IEEE Real-Time Systems Symposium (RTSS),* Dec 2003, pp. 286-297.

[8] M. Ding, D. Chen, K. Xing, and X. Cheng, "Localized Fault-Tolerant Event Boundary Detection in Sensor Networks," in *Proc. of 24th IEEE INFOCOM,* Mar 2005, pp. 902-913.

[9] J. Staddon, D. Balfanz, and G. Durfee, "Efficient Tracing of Failed Nodes in Sensor Networks," in *Proc. of 1st ACM International Workshop on Wireless Sensor Networks and Application (WSNA)*, Sep 2002, pp. 122-130.

[10] F. Ye, H. Luo, S. Lu, and L. Zhang, "Statistical En-Route Filtering of Injected False Data in Sensor Networks," in *Proc. of the 23rd IEEE INFOCOM,* Mar 2004, pp. 2446-2457.

[11] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks," *in Proc. of the 22th IEEE INFOCOM 2003*, April 2003, pp. 1976-1986.

[12] L. Lazos, R. Poovendran, C. Meadows, P. Syverson, and L.W. Chang, "Preventing Wormhole Attacks on Wireless Ad Hoc Networks: A Graph Theoretic Approach," in *IEEE Wireless and Communications and Networking Conference (WCNC)*, Mar 2005, pp.1193-1199.

[13] L. Hu and D. Evans, "Localization for Mobile Sensor Networks," in *Proc. of the 10th ACM MobiCom*, Sep 2004, pp. 45-57.

[14] D. Wagner, "Resilient Aggregation in Sensor Networks," in *ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN)*, Oct 2004, pp. 78-87.

[15] N. Ye and Q. Chen, "An Anomaly Detection Technique Based on a Chi-square Statistic for Detecting Intrusions into Information Systems," *Quality and Reliability Engineering International*, vol. 17, no. 2, 2001, pp. 105-112.

[16] D. Liu and P. Ning, "Establishing Pairwise Keys in Distributed Sensor Networks," in *Proc. of the 10th ACM Conference on Computer and Communications Security (CCS)*, Oct 2003, pp. 52-61.

[17] P. Ganesan, R. Venugopalan, P. Peddabachagari, A. Dean, F. Mueller, M. Sichitiu, "Analyzing and Modeling Encryption Overhead for Sensor Network Nodes," in *Proc. of the 2nd ACM International Conference on Wireless Sensor Networks and Applications (WSNA)*, Sep 2003, pp. 151-159.

[18] S. Zhu, S. Setia, S. Jajodia, "LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks," in *Proc. of the 10th ACM Conference on Computer and Communications Security (CCS)*, Oct 2003, pp. 62-72.

[19] W. B. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "An Application-Specific Protocol Architecture for Wireless Microsensor Networks," *IEEE Transactions on Wireless Communications*, vol. 1, no. 4, Oct 2002, pp. 660-670.

[20] "MPR/MID User's Manual," *Document 7430-0021-06*, Rev. B, Crossbow Technology, Inc., April 2005.

[21] R. Jurdak, C. V. Lopes, P. Baldi, "Battery Lifetime Estimation and Optimization for Underwater Sensor Networks," *IEEE Sensor Operations*, 2004.