Intrusion Detection of Sinkhole Attacks in Wireless Sensor Networks

Ioannis Krontiris, Tassos Dimitriou, Thanassis Giannetsos, and Marios Mpasoukos

Athens Information Technology,
P.O.Box 68, 19.5 km Markopoulo Ave.,
GR- 19002, Peania, Athens, Greece
{ikro,tdim,agia,mamp}@ait.edu.gr

Abstract. In this paper, we present an Intrusion Detection System designed for wireless sensor networks and show how it can be configured to detect Sinkhole attacks. A Sinkhole attack forms a serious threat to sensor networks. We study in depth this attack by presenting how it can be launched in realistic networks that use the MintRoute protocol of TinyOS. MintRoute is the most widely used routing protocol in sensor network deployments, using the link quality metric to build the corresponding routing tree. Having implemented this attack in TinyOS, we embed the appropriate rules in our IDS system that will enable it to detect successfully the intruder node. We demonstrate this in our own sensor network deployment and we also present simulation results to confirm the effectiveness and accuracy of the algorithm in the general case of random topologies.

1 Introduction

Most of the applications in wireless sensor networks (WSN) require the unattended operation of a large number of sensors. This fact along with the limited computational and communication resources of their nodes make them susceptible to attacks. Sensor networks cannot rely on human intervention to face an adversary's attempt to compromise the network or hinder its proper operation. Instead, an autonomic response of the network that relies on the embedded pre-programmed policies and a coordinated, cooperative behavior is the most effective way to gain maximum advantage against adversaries.

So far, research in sensor networks security has made certain progress in providing specialized security mechanisms, like key establishment [1], secure localization [2], or secure aggregation [3]. Also, security protocols have been designed with the goal of protecting a sensor network against particular attacks, like selective forwarding [4], sinkhole [5] or wormhole [6] attacks. However, all these protocols fall prey to *insider* attacks, in which the attacker has compromised and retrieved the cryptographic material of a number of nodes. Because of their resource constraints, sensor nodes usually cannot deal with such strong adversaries. So what is needed is a second line of defense: An *Intrusion Detection*

System (IDS) that can *detect* a third party's attempts of exploiting protocol weaknesses and *warn* of malicious behavior. Using an IDS, the network will be able to respond and isolate the intruder in order to protect and guarantee its normal operation.

In [7], we proposed an IDS for sensor networks which is designed to work with only partial and localized information available in each node. In particular, we concentrated on how such an IDS could detect *blackhole* and *selective forwarding* attacks. The nodes simply monitor their neighborhood and collaborate with each other sharing valuable information that eventually leads to the successful detection of the attack.

In this paper we extend this IDS system so that it can detect *sinkhole* attacks, a particularly severe attack that prevents the base station from obtaining complete and correct sensing data, thus forming a serious threat to higher-layer applications. By showing how the detection of such attacks can be integrated in the IDS, we move a step further towards a complete intrusion detection solution for sensor networks.

Current routing protocols in sensor networks are susceptible to sinkhole attacks [8]. This is because these protocols were not designed having security threats in mind. In this paper, we concentrate on MintRoute, which is among the most widely used routing protocols in TinyOS. MintRoute is used in most real sensor networks deployments today, as for example in [9,10,11], therefore it is important to guarantee protection of such networks from sinkhole attacks. To emphasize this further, we also demonstrate how easily it is for an intruder to launch a sinkhole attack against a network having MintRoute as its underlying routing protocol. We implemented both the attack and the IDS in TinyOS to demonstrate the effectiveness and accuracy of the intrusion detection process.

The remainder of this paper is organized as follows. Section 2 references related work. In Section 3, we review MintRoute emphasizing on its basic characteristics that an attacker could exploit to launch a sinkhole attack. In Section 4, we present in detail how this attack can be realized by an intruder node, and in Section 5, we present the architecture of our IDS system. Finally, in Section 6, we present simulation results that show the behavior of the IDS in several random topologies and our demonstration on a realistic sensor network deployment.

2 Related Work

There are currently only a few studies in the area of intrusion detection in wireless sensor networks. Da Silva *et al.* [12] and Onat and Miri [13] propose two similar IDS systems, where certain monitoring nodes in the network are functioning as watchdogs for their neighbors, looking for intruders. They listen to messages in their radio range and store in a buffer specific message fields that might be useful to an IDS system running within a sensor node, but no details are given how this system works. In these architectures, there is no collaboration among the monitor nodes. It is concluded from both papers that the buffer size is an important factor that greatly affects the rate of false alarms.

A first approach on the intrusion detection of sinkhole attacks has been presented in [5]. However this approach involves the base station in the detection process, resulting in a high communication cost for the protocol. Furthermore, it cannot be generalized to an IDS that could detect more types of attacks, as it is designed just for that particular attack. We believe that having a more general IDS architecture first can be enriched later with the support for more attacks leading to a complete solution.

3 Mintroute

MintRoute is one of the most commonly used routing protocol in TinyOS. It uses link quality estimates as the routing cost metric to build the routing tree towards the base station. For the calculation of these link estimates, each node periodically transmits a packet, called "*route update*".

Each node estimates the link quality of its neighbors based on the *packet loss* of the route update packets received from each corresponding neighbor. The list of these estimates for each neighbor is broadcasted by the node periodically in its own route update packets.

Every node maintains a Neighbor Table and updates it when it receives a route update packet. This table stores a list with the IDs of all neighboring nodes and their corresponding link costs. The node chooses its "parent node" to be the one with the best link quality in the Neighbor Table. Note that the hop distance of each neighbor to the base station is not taken under consideration in choosing the parent.

The parent changing mechanism is triggered each time the link quality of one or more nodes becomes 75% better than the link quality of the current parent, or the link quality of the current parent drops below 25 (in absolute value). In such case, the node with the highest quality becomes the new parent. However, if two of such candidate nodes happens to have the same link quality, the new parent will be the first one found in the Neighbor Table.

4 Sinkhole Attack

In a Sinkhole attack [8] a compromised node tries to draw all or as much as possible traffic from a particular area, by making itself look attractive to the surrounding nodes with respect to the routing metric. As a result, the adversary manages to attract all traffic that is destined to the base station. By taking part in the routing process, she can then launch more severe attacks, like selectively forwarding, modifying or even dropping the packets coming through.

A compromised node does not necessarily have to target other nodes from areas outside its neighborhood in order to control traffic. The adversary needs only to launch the sinkhole attack from a node as close as possible to the base station. In this case, by having the neighboring nodes choose the intruder as their parent, all the traffic coming from their descendants will also end up in the sinkhole. So the attack can be very effective even if it is launched locally, with small effort from the side of the attacker.

In the case of a routing protocol, like MintRoute, that uses link estimates as the routing metric, the compromised node launching the sinkhole attack will try to persuade its neighbors to change their current parents and choose the sinkhole node as their new one, by trying to make these parents look like they have much worse link quality than itself. Note that in this case of MintRoute, the attacker cannot launch a sinkhole attack by advertising that is has a lower hop count to the base station, as this metric is not used in the routing protocol. So the attacker needs to come up with more sophisticated ways.

Moreover, by just advertising a high link quality to the other nodes may not be enough to make them change their parents, since most of these routing protocols try to be robust, meaning that they don't allow the nodes change parents frequently and for no good reason. For example, when a node changes its parent, that could create a routing cycle in the network, which is followed by an extra cost to resolve it. Therefore, aside from advertising a high link quality for itself, the attacking node needs to make the current parents look like they have a very poor link quality, which will trigger the parent changing mechanism in their children. Then the new parent to be chosen will be the sinkhole node.

One sinkhole attack using the MintRoute as the underlying routing protocol is presented in [14]. The method of this attack is to change the link quality estimates sent by the nodes, within the route update packets. To do that, the attacker listens to the route update messages from its neighbors, alters them and replays them impersonating the original sender. Even if there is an underlying key mechanism which nodes can use to communicate securely with each other, most probably the attacker will be using a broadcast key shared with the nodes to be able to overhear change and send these packets.

Let's take for example the case shown in Figure 1, where node C is the attacker and node B is the current parent of node A. Node C has sent its own route update packets advertising a fake link quality (at the maximum value of 255), but this is not enough to make node A change its parent. Therefore, when it receives the route update packet of node A, it changes the link quality of node

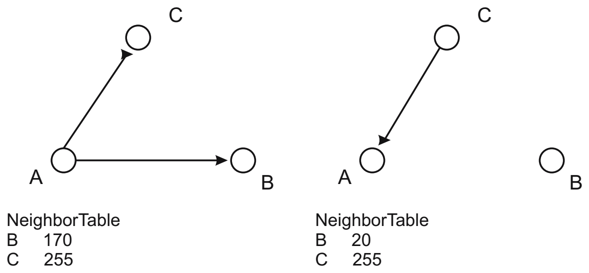


Fig. 1. The two phases of sinkhole attack. In the first phase node C (attacker) receives the route update packet of node A and in the second phase it sends the forged packet to A impersonating B.

B to a low value and sends it back to A as a unicast packet, impersonating B. Upon receiving this packet, node A thinks it is a route update packet from B and updates the corresponding entry in the Neighbor Table. This will trigger the parent changing mechanism and since the link quality of node B is below 25, that node will be ignored and node C will be chosen.

After performing the above attack for all of its neighbors, eventually the Sinkhole node will attract most (if not all) of the networks traffic.

5 Intrusion Detection

In this section we propose an IDS for sensor networks that is able to detect an ongoing sinkhole attack. For the design of our solution we have assumed a routing layer that is based on link quality metrics to form a routing tree towards the base station.

The intrusion detection system follows a distributed architecture. It is composed of identical IDS clients running in each node in the network. Then the IDS clients communicate with each other in order to reach a conclusion on an intrusion event. The functionality of each IDS client can be summarized as following:

– *Network Monitoring*: Each IDS client listens on the network and captures and examines individual packets passing from its immediate neighborhood in real time. Since all communication in a WSN is conducted over the air, and each node can overhear the traffic in its neighborhood, this is a natural audit source for the IDS client.
– *Intrusion Detection*: Each IDS client follows a specification-based approach in order to detect attacks, i.e., it detects deviations from normal behavior based on user defined rules. The network administrator have to define and embed in the motes the corresponding rules for each attack that the IDS should detect. In this paper we define the rules for the sinkhole attack, which we will present shortly.
– *Decision Making*: Due to its myopic vision around its neighborhood, a node may not be able to make a final decision whether a node is indeed an intruder. But even if it is, it cannot be trusted by the network, as it can be malicious itself. Therefore, if an anomaly is detected by an IDS client then a cooperative mechanism is initiated with the neighboring nodes so that all of them come to a mutual conclusion.
– *Action*: Every node has a response mechanism that allow it to respond to an intrusion situation.

Based on these functions we build the architecture of the IDS client around five conceptual modules, as shown in Figure 2. Each module is responsible for a specific function, which we describe in the sections below. The IDS clients are identical in each node and they can exchange messages with clients in neighboring nodes.

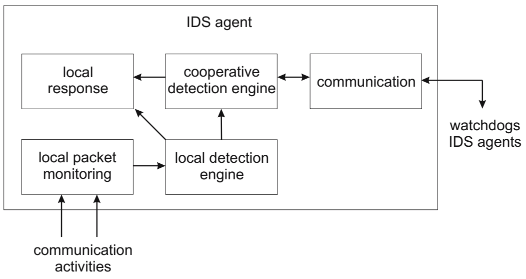


Fig. 2. The building blocks of the IDS client existing in each sensor node

5.1 Local Packet Monitoring

This module gathers audit data to be provided to the local detection module. Audit data in a sensor network IDS system can be the communication activities within its radio range. This data can be collected by listening *promiscuously* to neighboring nodes' transmissions. As sensor nodes have this capability, this can be very useful for intrusion detection.

In particular, in our IDS design we require that for each node in the network, any of its neighbors listening to the packets that this node is sending or receiving will participate in the intrusion detection procedure. Therefore, the neighbors of a node function as watchdogs for that node. As we will see in Section 5.3, one watchdog is *not enough* to detect a sinkhole attack, but if all neighbors contribute their point of view to the rest of them, then the picture becomes complete, and the attacker is revealed.

More importantly, there is no need for the watchdogs to store the overheard packets or any other information in their memory. It is just enough to temporarily buffer each packet in order to apply the rules defined by the local detection engine and see if any of these rules are satisfied. Then the packet can be discarded. No historical or statistical data need to be kept in the node's memory.

5.2 Local Detection Engine

The local detection engine stores and applies all the specifications that describe what is a correct operation and monitors audit data with respect to these constraints, in order to identify any deviations from normal behavior. These specifications are defined in the form of rules, specified by the developer, since we want to avoid the overhead of training the network to what is a normal behavior.

In order to detect the sinkhole attack we add two rules that will trigger an alert whenever the malicious node tries to impersonate another node, according to the attack we described in Section 4. The intuition is that route update packets should originate only from their legitimate sender and the nodes should defend against impersonation attacks.

Rule 1: *"For each overhead route update packet check the sender field, which must be different than your node ID. If this is not the case, produce an alert and broadcast it to your neighbors."*

Rule 2: *"For each overhead route update packet check the sender field, which must be the node ID of one of your neighbors. If this is not the case, produce an alert and broadcast it to your neighbors."*

For a node that detects an anomaly according to the above rules it is only an indication that a sinkhole attack is in process. However there is no way to know which node is trying to launch the attack, since the sender field is altered. The only conclusion that can be drawn so far is that the attacker is one of the neighboring nodes, since the route update packets are only broadcasted locally. So, we need to rely on the cooperation of the nodes to reduce the candidates to one node, i.e. the attacker.

5.3 Cooperative Detection Engine

The problem we need to solve is how the intruder node will be revoked from the network. Basing this decision on an individual node is not sufficient for two reasons:

1. The node who makes the final decision can be compromised itself. Then it could choose not to revoke an attacking node or revoke a legitimate one. So, the decision should be collaborative, and should come from all the nodes that are involved, i.e. the watchdogs.
2. In the case of sinkhole attack there is not enough information in only one node to conclude on the attacker.

Therefore, we need a cooperative detection engine that will guide the nodes through a safe conclusion on which node is the intruder so that it can be revoked by the network. In particular, we are going to exploit the fact that when a malicious node launches a sinkhole attack, one of the two rules in the local detection engine will be triggered at several of its neighbors. If these neighbors collaborate, it turns out that they can identify the attacker.

The collaborative approach consists of having each watchdog node broadcasting its neighbors list. As we said, each watchdog that produces a local alert can conclude nothing more than that the attacker is one of its neighbors. However, if all the nodes that produce the alert communicate their neighbors to each other, the attacker has to be one of the nodes in the *intersection* of these sets.

So, to set it more formally, if there is evidence of intrusion produced at the local detection engine, the cooperative detection module broadcasts an alert to the neighboring nodes. The alert is composed by the list of the IDs of the sender's *neighbors*. Upon receiving such an alert, and provided that it is a watchdog itself, a node excludes from the potential attackers all the node IDs in the alert that are not part of its own neighbor list. Thus it performs an intersection between its own neighbor list and the node list found in the alert. The result will be

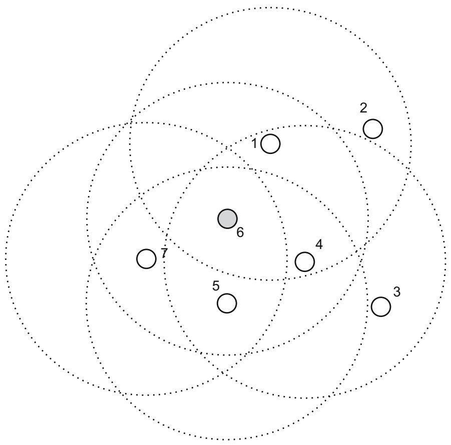


Fig. 3. An example topology where node 6 is the attacker. If each node broadcasts an alert with the list of its neighbors, the intersection of the alerts each node receives is node 6.

stored and used for the intersection with the next alert that the node is going to receive. Thus each time a watchdog is receiving an alert, the intersection will give an even smaller set of nodes. If at the end there is only one node left at the result, that node is the attacker.

The intuition in this is that each time a node broadcast an alert with its neighbors, this set is a set that includes the attacker. By exchanging these sets and performing the intersections, nodes are looking to find which nodes are common within the sets. If some nodes manage to reduce this set down to one node, then they can be sure about the intruder's identity. Let's see this in an example.

In Figure 3, the attacker is node 6. Each node broadcasts an alert including its neighbor list. Then, it intersects the lists from the alerts it receives along with its own neighbor list. So, we will have the following results:

Node 1: $\{2, 4, 6\} \cap \{1, 3, 5, 6\} = \{6\}$
Node 4: $\{1, 3, 5, 6\} \cap \{2, 4, 6\} \cap \{4, 6, 7\} = \{6\}$.
Node 5: $\{4, 6, 7\} \cap \{1, 3, 5, 6\} \cap \{5, 6\} = \{6\}$.
Node 7: $\{5, 6\} \cap \{4, 6, 7\} = \{6\}$

Each node remains with a set of one node, the attacker. It could be the case (not demonstrated in the example) that the set had more than one node and then no conclusion could be drawn. However, we will see in the experimental section that more than 75% of the attacker's watchdogs will manage to successfully detect it, meaning that they will end up with only one node ID as the result of the cooperative detection process.

5.4 Local Response

Once the watchdogs are aware that an intrusion has taken place and have detected the compromised node, appropriate actions are taken by the local response module. The first action is to cut off the intruder as much as possible and isolate the compromised node. After that, proper operation of the network must be restored. This may include changes in the routing paths, updates of the cryptographic material (keys, etc.) or restoring part of the system using redundant information distributed in other parts of the network. Autonomic behavior of sensor networks means that these functions must be performed without human intervention and within finite time.

Depending on the confidence and the type of the attack, we categorize the response to two types:

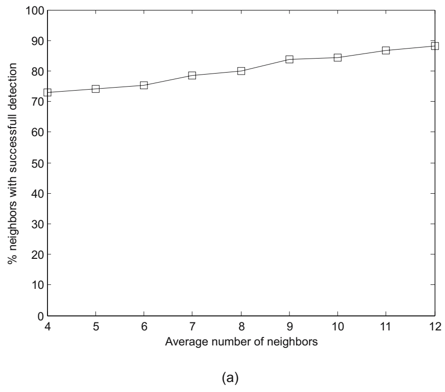
– *Direct response*: Excluding the suspect node from any paths and forcing regeneration of new cryptographic keys with the rest of the neighbors.
– *Indirect response*: Notifying the base station about the intruder or reducing the quality estimation for the link to that node, so that it will gradually loose its path reliability.

We will not go in more depth regarding the local response process, since for this paper we are more concentrated on the former steps of the intrusion detection. However, let us note that in any case, the local response cannot be based on the claims of one of a few nodes, since the attacker may have compromised and use more than one node during the attack. Therefore, we require that the majority of the attacker's neighbors have successfully detect the sinkhole node. Otherwise we count the case as a false negative, i.e. the network could not reach a safe conclusion. We measure the false negative rate in the experimental section that follows.

6 Experimental Evaluation

We have simulated a sensor network of 100 nodes placed uniformly at random in order to test our proposed intrusion detection system. For each run of the simulation, we chose at random one node to launch a sinkhole attack. This way we could have the watchdogs of that node apply the intrusion detection and monitor its behavior.

To measure the success of the IDS system on identifying the intruder, we first run the simulation 1000 times and produced the average number of the watchdogs that ended up with only one ID (the intruder ID) as the result of the alerts intersection. As we see in Figure 4(a), the majority of the watchdogs were able to detect the malicious node. Let's also note that as the network density increases, the results improve, meaning that bigger portion of the watchdogs manage to identify the attacker. This is due to the fact that more watchdogs are within the range of each other, so the intersection of the alerts is more likely that it will produce a unique node ID. For example, for a network density of



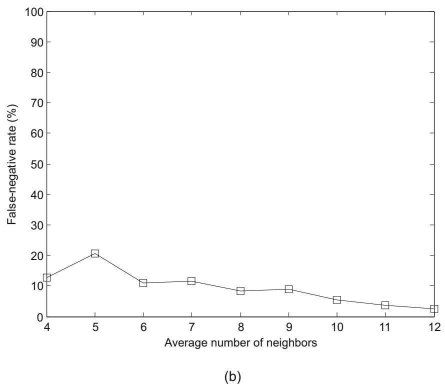


Fig. 4. (a) The percentage of watchdogs that successfully detect the attacker for different network densities. (b) False-negative rate for different network densities.

6 neighbors on average, 75% of them will identify the attacker, while for 12 neighbors, the percentage goes up to 88.3%.

Next we measured the false negative rate of our IDS system. We define a false negative as the case where the majority of the watchdogs were not able to conclude to one node ID and therefore identify the attacker. This is possible if the topology is such that the intersection of the alerts that each watchdog receives produces a set of more than one node. However, as we see in Figure 4(b), the probability that less than half of the watchdogs remain inconclusive is very low and becomes even lower as the network density increases. For example, for a network density of 6 neighbors on average, the false negative rate is 11%, while for 10 neighbors it is 5.3%.

Next we implemented our IDS system in TinyOS in order to evaluate it in a realistic deployment of sensor nodes (Mica2) and analyze its memory requirements per node. In particular, it required 1.5 KB in RAM (out of 4 KB available) and 3.9 KB in ROM (out of 128 KB available), which are realistic memory overheads for an intrusion detection system. However we are not aware of any other IDS implementation in TinyOS to use as a comparative measure.

We programmed a node (node 5) to launch the sinkhole attack as described in Section 4. Then we programmed the rest of the nodes with MintRoute so that they could form a routing tree as shown in Figure 5. We also wired our IDS client in each of these nodes to see if they could detect the attack and identify the intruder node.

Indeed, nodes $1, 4, 6$ and 8 produced intrusion alerts and successfully identified node 5 as the attacker. Let's consider node 8 for example. Its neighbors are nodes $\{1, 4, 5, 6\}$. The attacker tried to impersonate it, sending routing update packets to its children, node 4, so an alert was produced due to rule 1, as we described in Section 5.2. Moreover, it received the alerts from nodes $4, 8$ and 1, which

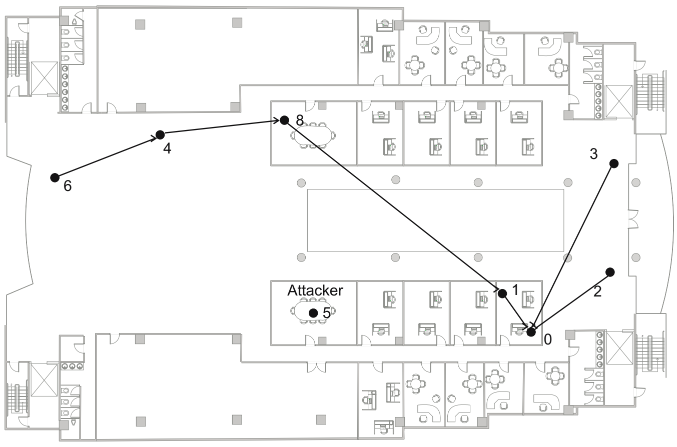


Fig. 5. The routing tree formed by the sensor nodes using MintRoute. Node 5 is the attacker launching a sinkhole attack.

also produced alerts when the attacker tried to attract the rest of the nodes. The intersection of these alerts with its own neighbors resulted in just node 5: $\{1,4,5,6\} \cap \{4,8,5\} \cap \{8,6,5\} \cap \{0,2,3,5,8\} = \{5\}$.

Similarly, for the rest of the nodes we had

Node 1: $\{0,2,3,5,8\} \cap \{4,1,0,5,6\} = \{5\}$
Node 4: $\{8,6,5\} \cap \{4,8,5\} \cap \{4,1,0,5,6\} = \{5\}$
Node 6: $\{4,8,5\} \cap \{4,1,0,5,6\} \cap \{8,6,5\} = \{5\}$

Let us note that the attacker's neighbors are $\{1,2,4,6,8\}$. Therefore, 4 out of its 5 neighbors successfully detected the attack. No rules were triggered for nodes 2 and 3, so they remained out of the process.

7 Conclusions

In this paper, we described a model for a distributed intrusion detection system that uses a large number of autonomous, but localized, cooperating agents in order to detect a node launching a sinkhole attack. The nodes use coordinated surveillance by incorporating inter-agent communication and distributed computing in decision making to identify characteristic signs of the attack, and raise an appropriate alarm. In particular, we concentrated on the sinkhole attack against routing update protocols based on link quality like MintRoute and we described the appropriate specifications that need to be implemented by the IDS system so that it can detect such attacks. We believe this set of principles can be used as a valuable tool for developing more robust and secure sensor networks in the future and enable further research in the area.

References

1. Camtepe, S., Yener, B.: Key distribution mechanisms for wireless sensor networks: a survey. Technical Report 05-07, Rensselaer Polytechnic Institute, Troy, New York (March 2005)
2. Lazos, L., Poovendran, R.: Serloc: Robust localization for wireless sensor networks. ACM Transactions on Sensor Networks 1(1), 73–100 (2005)
3. Dimitriou, T., Krontiris, I.: Secure In-network Processing in Sensor Networks. In: Security in Sensor Networks, pp. 275–290. CRC Press, Boca Raton, USA (2006)
4. Yu, B., Xiao, B.: Detecting selective forwarding attacks in wireless sensor networks. In: Proceedings of the 20th International Parallel and Distributed Processing Symposium (SSN2006 workshop), Rhodes, Greece, pp. 1–8 (April 2006)
5. Ngai, E.C.H., Liu, J., Lyu, M.R.: On the intruder detection for sinkhole attack in wireless sensor networks. In: ICC 2006. Proceedings of the IEEE International Conference on Communications, Istanbul, Turkey (2006)
6. Hu, Y.C., Perrig, A., Johnson, D.B.: Packet leashes: A defense against wormhole attacks in wireless ad hoc networks. In: INFOCOM 2003. Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies, San Francisco, CA, USA (2003)
7. Krontiris, I., Dimitriou, T., Freiling, F.C.: Towards intrusion detection in wireless sensor networks. In: Proceedings of the 13th European Wireless Conference, Paris, France (April 2007)
8. Karlof, C., Wagner, D.: Secure routing in wireless sensor networks: Attacks and countermeasures. AdHoc Networks Journal 1(2–3), 293–315 (2003)
9. Baggio, A.: Wireless sensor networks in precision agriculture. In: REALWSN 2005. Proceeding of the Workshop on Real-World Wireless Sensor Networks, Stockholm, Sweden (June 2005)
10. Werner-Allen, G., Lorincz, K., Welsh, M., Marcillo, O., Johnson, J., Ruiz, M., Lees, J.: Deploying a wireless sensor network on an active volcano. IEEE Internet Computing 10(2), 18–25 (2006)
11. Schmid, T., Dubois-Ferrière, H., Vetterli, M.: SensorScope: Experiences with a Wireless Building Monitoring Sensor Network. In: REALWSN 2005. Proceeding of the Workshop on Real-World Wireless Sensor Networks, Stockholm, Sweden (June 2005)
12. da Silva, A.P., Martins, M., Rocha, B., Loureiro, A., Ruiz, L., Wong, H.C.: Decentralized intrusion detection in wireless sensor networks. In: Q2SWinet 2005. Proceedings of the 1st ACM international workshop on Quality of service & security in wireless and mobile networks, pp. 16–23. ACM Press, New York (2005)
13. Onat, I., Miri, A.: An intrusion detection system for wireless sensor networks. In: Proceeding of the IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, Montreal, Canada, August 2005, vol. 3, pp. 253–259 (2005)
14. Datema, S.: A case study of wireless sensor network attacks. MSc thesis, Delft University of Technology (2005)