Cooperative Intrusion Detection
in Wireless Sensor Networks

Ioannis Krontiris[1], Zinaida Benenson[2,*], Thanassis Giannetsos[1],
Felix C. Freiling[2], and Tassos Dimitriou[1]

[1] Athens Information Technology, 19.5 Km Markopoulo Avenue, Peania, Greece
`{ikro,tdim,agia@ait.edu.gr}`
[2] Laboratory for Dependable Distributed Systems, University of Mannheim,
68131 Mannheim, Germany
`zina@uni-mannheim.de, freiling@informatik.uni-mannheim.de`

Abstract. We consider the problem of cooperative intrusion detection
in wireless sensor networks where the nodes are equipped with local de-
tector modules and have to identify the intruder in a distributed fashion.
The detector modules issue suspicions about an intrusion in the sensor's
neighborhood. We formally define the problem of intrusion detection and
identify necessary and sufficient conditions for its solvability. Based on
these conditions we develop a generic algorithm for intrusion detection
and present simulations and experiments which show the effectiveness of
our approach.

Key words: sensor networks, security, intrusion detection

1 Introduction

The pervasive interconnection of autonomous and possibly wireless sensor de-
vices has given birth to a broad class of exciting new applications in several areas
of our lives, including environment and habitat monitoring, healthcare applica-
tions, home automation, and traffic control. At the same time, however, their
unattended nature and the limited resources of their sensor nodes have created
an equal number of threats posed by attackers in order to gain access to the
network and the information transferred within.

There are several classical security methodologies so far that focus on trying
to *prevent* these intrusions. A lot of work in sensor network security has focused
on particular types of attacks and how they can be prevented. This can, however,
only be a first line of defense. It is impossible, or even infeasible, to guarantee
perfect prevention. Not all types of attacks are known and new ones appear
constantly. As a result, attackers can always find security holes to exploit. For
certain environments it makes sense to establish a second line of defense: An
Intrusion Detection System (IDS) that can *detect* an attack and *warn* the sensors
and the operator about it.

1.1   Related Work

Intrusion detection has received some attention in wireless sensor networks before. Most work has focused on *local detection*, i.e., allowing nodes to locally detect specific attacks which are performed in their neighborhood [2, 10, 4, 5, 3, 8, 1].

Da Silva et al. [2] and Onat and Miri [10] propose similar IDS systems, where certain monitor nodes in the network are responsible for monitoring their neighbors. 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. Kargl et al. [3] focus on the detection of *selfish* nodes that try to preserve their resources at the expense of others. Loo et al. [8] and Bhuse and Gupta [1] describe two more IDSs for sensor networks. Both papers assume that routing protocols for ad hoc networks can also be applied to WSNs.

In all the above work, there is no collaboration among the sensor nodes. The only collaborative approaches we are aware of focus on the local detection of selective forwarding attacks [4] and sinkhole attacks [5].

More extensive work has been done in intrusion detection for ad hoc networks [9]. In such networks, distributed and cooperative IDS architectures are also preferable. Detailed distributed designs, actual detection techniques and their performance have been studied in more depth. While also being ad hoc networks, wireless sensor networks are much more resource constrained. We are unaware of any work that has investigated the issue of intrusion detection in a general collaborative way for wireless sensor networks.

1.2   Contributions

In this paper we study a general approach to intrusion detection in wireless sensor networks which focuses more on the collaboration of sensors than on the detection of specific attacks. We believe that wireless sensor networks with their inherent redundancy are ideal for applying cooperative techniques. We therefore abstract from concrete local detection techniques and define in Section 2 the notion of a local *alert module*. Such modules issue suspicions about an intrusion in the sensor's neighborhood. We focus here on the case where sensor nodes try to detect a *single* malicious node, as this case is already surprisingly complex.

We formally define the problem of intrusion detection in Section 3 and identify necessary and sufficient conditions on the behavior of the local alert modules such that they contain enough information to cooperatively solve the intrusion detection problem in Section 4. These conditions also identify scenarios in which cooperative intrusion detection is unsolvable.

We further develop an algorithm that solves intrusion detection based only on the output of the alert modules in Section 5. The idea is that nodes in the neighborhood of the attacker exchange information about who they suspect and jointly identify the attacker. Note that this is not an easy task since the attacker can also participate in the protocol and try to bring honest nodes to a wrong conclusion. In this sense, intrusion detection may at first sight seem similar to

the problem of Byzantine agreement [11]. However, we show that both problems are incomparable.

Finally in Section 6 we investigate the probability that symmetry conditions that make intrusion detection impossible to solve occur in practice, using simulations. In Section 7, we present a lightweight implementation of our cooperative intrusion detection algorithm on Moteiv Tmote Sky motes justifying the practicality of our approach.

2   System Model

2.1   Sensor Nodes and Communication

We present a strong system model used for proofs of necessary and sufficient conditions for intrusion detection in the sequel. It is useful, because if some problem is impossible to solve in our model, it is also impossible to solve in weaker models which are closer to the reality.

In our model, a wireless sensor network consists of a set $S = \{s_1, s_s, \ldots, s_n\}$ of n sensor nodes. Sensors communicate by sending messages over a wireless broadcast medium, meaning that if some sensor s sends a message, many other sensors can receive the message simultaneously. Possible collisions on the wireless medium can only delay the receipt of a message for a relatively short time.

For any sensor node s, the set of nodes with which it can directly communicate is denoted by $N(s)$. For simplicity, we assume a static and symmetric neighborhood relation, i.e., if $s \in N(s')$ then $s' \in N(s)$. We assume that every node knows its 2-hop-neighborhood.

Although the above assumptions are strong, considering unreliable asymmetric wireless links and frequent neighborhood changes in real sensor networks, we use them especially for proofs. In Section 4.4 we discuss how the system model can be weakened. We also present an implementation of our algorithms for real sensor networks in Section 7 which does not make these strong assumptions.

We make no assumptions about the communication topology defined by all neighborhood sets apart from that all nodes that behave according to the protocol (honest nodes) are connected via a path consisted only of other honest nodes. We expect that in typical sensor networks the density of the nodes is rather high so that this condition will be satisfied with high probability.

2.2   Attacker Model

We assume that an attacker can capture at most t nodes to launch an attack on the sensor network. We model this by allowing at most t nodes to behave in an arbitrary manner (Byzantine failure). However, we do not propose a Byzantine Agreement protocol, but focus on the Intrusion Detection Problem (Definition 3). The relationship between Intrusion Detection and Byzantine Agreement is shown in Section 4.3.

In the following, we concentrate on the case where $t = 1$. In this case, we call the captured node the *source* of the attack, or the attacker, and use the

predicate $source(s)$ which is true if and only if (iff) s is the attacker. All other nodes are called honest: $honest(s) \equiv \neg source(s)$. As the rigorous examination of this case already gives very useful insights on solvability of intrusion detection (and turns out to be quite complex), we leave the case $t > 1$ to future work.

2.3   Alert Module

Attacks are locally detected by a local intrusion detection mechanism. We abstract such mechanisms of a sensor node into an *alert module*. Whenever the alert module at node s notices something wrong in its neighborhood, the alert module outputs some set $D(s)$ of *suspected sensors*, called the *suspected set*. If $|D(s)| = 1$, then the node has identified the attacker. Most often however, $D(s)$ will contain a larger set of neighbors or may even be equal to $N(s)$.

For example, in the detection of the selective forwarding attack [4], the nodes are able to identify the one node that drops the messages by monitoring the transmissions in their neighborhood ($|D(s)| = 1$). On the other hand, in the protocol for detecting the sinkhole attack [5] nodes only know that the attacker has to be one of their neighbors ($D(s) = N(s)$).

Formally, the alert module satisfies the following properties:

- If the alert module at a honest node s outputs $D(s)$, then the source is in that set, i.e., $\exists s' \in D(s) : source(s')$.
- If the attacker attacks, then within some time delay δ the alert module at some sensor s outputs a set $D(s)$.
- Only neighbors are suspected by honest nodes, i.e., $\forall s \in S : honest(s) \Rightarrow D(s) \subseteq N(s)$.

If the alert module at some node s outputs some set, we call s an *alerted node*. The predicate A on S denotes the set of alerted nodes, i.e., $A(s)$ holds iff s is an alerted node. The set of alerted nodes is called the *alerted set*. Note that the attacker may or may not belong to the alerted set, depending on the strategy that the attacker chooses to follow. Also, we do not require that *all* neighbors of the attacker belong to the alerted set.

3   The Intrusion Detection Problem

The cooperative intrusion detection process is triggered by an attack and by the subsequent alerts by the local alert modules of the neighboring sensors. The process ends by having the participating sensors jointly *expose* the source.

More formally, the predicate $expose_s(s')$ is true if node s exposes node s'. The intrusion detection problem can now be defined as follows:

Definition 1 (Intrusion Detection Problem (IDP)). *Find an algorithm that satisfies the following properties:*

- *(Correctness) If an honest node s exposes a node s', then s is in the alerted set and s' is the source, i.e., $\forall s \in S : honest(s) \wedge expose_s(s') \Rightarrow A(s) \wedge source(s')$.*

   – *(Termination) If the attacker attacks, then at most after some time τ all honest nodes in the alerted set expose some node.*

4   Conditions for Solving Intrusion Detection

The idea of cooperative intrusion detection is to exchange the outputs of local alert modules, thereby narrowing down the set of possible nodes that could be the attacker. In the following we assume that nodes have no other way to learn anything about the attacker than using their alert modules. As an initial example, consider the case depicted in Fig. 1(a). Node p suspects the source q, i.e., $D(p) = \{q\}$. Node q can claim to output $D(q) = \{p\}$. But p implicitly knows that it is honest, so it will ignore the information provided by q and expose q.
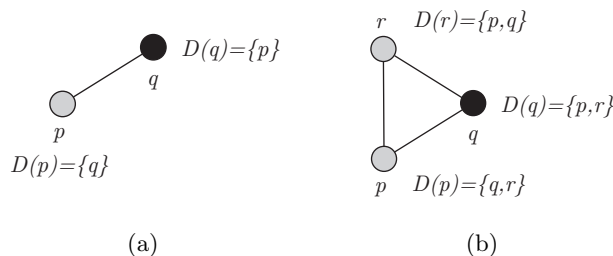

Fig. 1. Different types of alerted neighborhoods. Sources of attacks are marked black.

    In the example in Fig. 1(b), however, three nodes p, q, and r all suspect each other (node q is the source). As every node occurs in exactly two suspect sets, p cannot distinguish node r from node q by using only the suspect sets. We conclude that it is impossible to solve IDP in this case.

    Generalizing these two examples, the question arises about general conditions for the solvability of the intrusion detection problem. In the following, we give necessary (Section 4.2) and sufficient (Section 4.1) conditions for solvability of IDP using a deterministic algorithm for $t = 1$. We show the relationship between IDP and Byzantine Agreement in Section 4.3. In Section 4.4 we consider how weaker system models (unreliable links, neighborhood changes) affect the solvability of IDP.

4.1   Sufficient Conditions for Solving IDP

The Intrusion Detection Condition (IDC) We now give a sufficient condition for IDP solvability for $t = 1$ and deterministic algorithms. The intuition behind the condition is a generalization of the observation made in Fig. 1(b): If the suspected sets about some node s are structurally equivalent to those of the source, then the problem is in general not solvable.

Formally, for a node s we define the set $AN(s)$ to be the set of *alerted neighbors* of s, i.e.:

$$AN(s) = \{t|A(t) \wedge t \in N(s)\}$$

Furthermore, we define the set of alerted neighbors of p with respect to q $\tilde{AN}(p,q)$ to be the set of alerted neighbors of p without q, i.e.:

$$\tilde{AN}(p,q) = AN(p) \setminus \{q\}$$

For example, in Fig. 1(b) all three nodes are in alert mode and $AN(s) = D(s)$. The value of $\tilde{AN}(p,q) = \{r\}$ is the information content of $AN(p) = \{q,r\}$ that is valuable to q.

Definition 2 (Intrusion Detection Condition (IDC)). *The intrusion detection condition (IDC) is defined as:*

$$\forall p,q \in S : source(q) \Rightarrow \tilde{AN}(p,q) \neq \tilde{AN}(q,p)$$

Roughly speaking, IDC means that no other node has the same alerted neighborhood as the attacker. Note that if p and q are not neighbors, then IDC simplifies to:

$$\forall p,q \in S : source(q) \Rightarrow AN(p) \neq AN(q)$$

Theorem 1 (Sufficiency of IDC). *If $t = 1$, IDC is sufficient for ID, i.e., if IDC holds then IDP can be solved.*

Proof. Let all alerted nodes exchange their suspected sets. This is possible in our system model because each pair of honest nodes is connected by a path consisting of honest nodes, and communication is reliable.

Note that the attacker also can go into alert mode. Moreover, it can send different suspected sets to different nodes. However, as we assume that all nodes know their 2-hop-neighborhood, the suspected set of the attacker may only contain its neighbors. Otherwise, the attacker's messages would be discarded.

Consider the suspected sets received by an honest node p. If some node is suspected by a majority of the nodes, it is immediately identified as the attacker, because the attacker is included in the suspected set of every honest node.

A more complicated case arises when there are two or more nodes which are suspected by the majority of nodes. This situation can arise, e.g., if the attacker also goes into the alert mode and accuses some of its neighbors.

We denote the attacker as q. Assume that there is a node $p \neq q$ which is suspected by the same number of nodes as q. How can a node r distinguish between q and p?

(1) If $p = r$, then r knows that it is honest, and exposes q.

(2) Consider $p \neq r$. If all honest nodes suspect p, then the IDC does not hold. Thus, for some honest node s holds: $p \notin D(s)$ and $q \in D(s)$. It follows that q is alerted and $p \in D(q)$, as the number of nodes which suspect p is the same as the number of nodes which suspect q.

Node r must now decide which of nodes s and q lies about their suspicion. We now show that there is an alerted node v which is not neighbor of s. Indeed, if all alerted nodes were neighbors of s, then s and q would have the same alerted neighborhood with respect to each other, which contradicts the IDC. Thus, node r has to find out which of the nodes s and q is not a neighbor of some alerted node. This is possible as all nodes know their 2-hop neighborhood. This node has to be honest, and the remaining node is identified as the attacker.   □

As an example, consider Figure 2. Nodes s and r are honest nodes and alerted. Node p is also honest, but not alerted. The attacker is node q, which is alerted. In this example, nodes p and q are both suspected by two nodes. How can node r distinguish the attacker? IDC holds here, and node p is suspected by each of q and r. Thus, either q or s is lying about their suspicions. However, nodes r and s are not neighbors, and therefore, s cannot be the attacker.
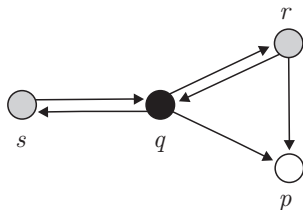

Fig. 2. Node q is the attacker, nodes s, r and q are alerted, while p is not alerted and it is marked white. $x \rightarrow y$ means that node x suspects node y. $D(r) = \{q, p\}$, $D(q) = \{p, r, s\}$, and $D(s) = \{q\}$.

The Neighborhood Conditions (NC) What happens if IDC is not satisfied? Can IDP still be solved, or is IDC also a necessary condition for solving IDP?

In the following we show that IDC is not a necessary condition. We give another sufficient condition for IDP solvability which can be valid in the network independently of the validity of the IDC.

Definition 3 (Neighborhood Conditions (NC)). *The Neighborhood Conditions (NC) consist of two conditions:*

- NC$_1$. *All neighbors of the attacker are alerted.*
- NC$_2$. *If two or more nodes are suspected by the majority of nodes, then all honest nodes suspected by the majority have non-alerted neighbors.*

Theorem 2 (Sufficiency of NC). *If the NC holds, then the IDP can be solved.*

Proof. We give an informal reasoning here. Let all alerted nodes exchange their suspected sets. If only one node is suspected by the majority of nodes, then this

node is the attacker, as all neighbors of the attacker are alerted (NC_1). If there are two or more nodes which are suspected by the majority, the nodes in the alerted set have to find out which of these nodes have non-alerted neighbors. According to NC_2, only the attacker does not have non-alerted neighbors.   □

4.2   Necessary and Sufficient Conditions for Solving IDP

We now show that for the solvability of IDP either the IDC or the NC (i.e., NC_1 and NC_2) should be satisfied in the sensor network.

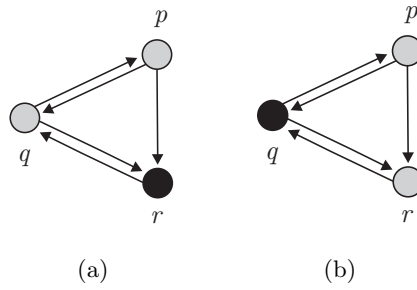**Theorem 3.** *IDP can be solved using a deterministic algorithm if and only if IDC or NC holds.*

(a)   (b)

Fig. 3. Case (a): Node p suspects q and r, node q suspects p and r, node r is the attacker and suspects q. IDC and NC_2 are not satisfied. Case (b): The suspicions remain as in case (a), but node q is the attacker. No algorithm for solving the IDP can distinguish between (a) and (b). Therefore, it is impossible to expose the attacker.

Proof. As shown in Theorems 1 and 2, if IDC holds or if NC holds, then the intrusion detection problem can be solved (sufficiency). We now show that it is also necessary for the solvability of the IDP that the IDC *or* the NC holds. It suffices to show that if the IDC does not hold and the NC does not hold, then the IDP cannot be solved.

Assume that the above claim is not true. That is, there exists a deterministic algorithm $\mathcal{A}$ that always exposes the attacker in case both the IDC and the NC do not hold. Consider Figure 3(a). The IDC does not hold there because $\tilde{AN}(p,r) = \tilde{AN}(r,p) = \{q\}$. Also NC does not hold, because NC_2 does not hold: The attacker r and the honest node q are suspected by two nodes, but q does not have non-alerted neighbors. In this case, the algorithm $\mathcal{A}$ should expose r. However, the situation in Figure 3(b) is exactly the same as in (a) from $\mathcal{A}$'s point of view. The suspicions remain the same, the topology also does not change. Thus, there is no additional information to help $\mathcal{A}$ to distinguish between situations (a) and (b). However, $\mathcal{A}$ should be able to distinguish between (a) and (b) and to expose r or q accordingly. It follows that $\mathcal{A}$ does not exist.   □

4.3   Byzantine Agreement vs. Intrusion Detection

In IDP, the honest nodes have to jointly expose the attacker. That is, they have to reach agreement on the attacker's identity. Although this looks similar to Byzantine Agreement [11], these two problems cannot be reduced to each other. In some cases, Byzantine Agreement can be solved whereas Intrusion Detection is not solvable, and vice versa.
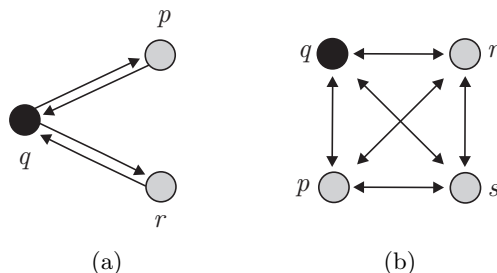

Fig. 4. Byzantine Agreement and Intrusion Detection cannot be reduced to each other. Case (a): Honest nodes p and r both suspect only the attacker q, thus Intrusion Detection can be solved, but Byzantine Agreement is not solvable. Case (b): Intrusion detection cannot be solved, Byzantine Agreement is solvable.

Consider Figure 4(a). Here, node q is the attacker and suspects both p and r. The honest nodes, on the other hand, both suspect q. In this case, Intrusion Detection is trivially solvable. However, Byzantine Agreement for three participants with $t = 1$ cannot be solved [11]. In Figure 4(b) all nodes suspect each other. IDC does not hold for nodes s and q, NC also does not hold, as no node has non-alerted neighbors. Thus, Intrusion Detection is not solvable. However, Byzantine Agreement for $t = 1$ can be solved here [11].

4.4   Solving IDP in a Weaker System Model

In proofs we used our assumptions on reliable and timely communication. In principle, we can also use weaker system models as long as they allow some protocols for reliable and timely exchange of suspected sets with high probability. For example, in our implementation we use an advertise-request protocol.

We also assumed a static and symmetric neighborhood relation. This assumption can also be weakened. All we need is that the nodes have secure information on their 2-hop neighborhood which does not change during a particular protocol run. In our implementation we let the nodes to find out their neighborhood in the secure initialization phase, where the attacker is absent. The neighborhood tables which are used for intrusion detection are then fixed. In case that a neighbor crashes, it looks in the protocol as if the node was not alerted. This can be

tolerated as long as the IDC holds (the NC does not hold in this case). On the other hand, if some new neighbors arrive, they can be ignored. This is problematic, however, in case the new node is the attacker. A better solution would be to have a secure protocol for the neighborhood table update. We leave this to future work.

5   A Cooperative Intrusion Detection Algorithm

Based on the ideas of Section 4, we now develop a general algorithm to solve the intrusion detection problem, i.e., all honest and alerted neighbors of an attacker share their partial views, agree on the identity of the source and expose it.

5.1   Initialization Phase

Prior to the deployment, each node is preloaded with a one-way key chain of length l, using a pre-assigned unique secret key K_l. A one-way key chain [7] $(K_0, K_1, \ldots, K_{l-1}, K_l)$ is an ordered list of cryptographic keys generated by successively applying a one-way hash function F (in our case SHA-1) to the key seed K_l, such as $K_j = F(K_{j+1})$, for $j = l - 1 \ldots 0$. Therefore, any key K_j is a commitment to all subsequent keys K_i, $i > j$; more specifically, K_0 is a key chain commitment for the entire chain. The length of the key chain l is a system parameter. In our implementation, we store the key chain on the external flash memory, such that it does not affect the memory requirements of our algorithm. This also allows us to set l to a large number, such that we can avoid the overhead of renewing the key chain during deployment.

The initialization phase takes place right after the network deployment. The duration of this phase is short enough so that we assume the absence of the attacker. We also require that all nodes discover their immediate neighbors, which is a standard procedure after deployment in almost all routing protocols. Further, all nodes discover their 2-hop neighborhood by broadcasting their IDs with a packet that has a TTL field equal to 2. The discovered neighborhood information is stored in the 2-hops neighborhood table. Then, each node announces their key chain commitment K_0 to all its 1-hop and 2-hop neighbors.

5.2   Voting Phase

During the voting phase each node in the alert region sends its vote to all the other members and respectively collects their votes. Let us denote the vote message from node s as $m_v(s)$. Each vote consists of the nodes suspected by the sender, so for node s, $m_v(s) = id||D(s)$. Node s "signs" its vote calculating the MAC with the next key K_j from its one-way key chain, and broadcasts

$$m_v(s), MAC_{K_j}(m_v(s)).$$

Following that, it sets a timer T_v to expire after time τ_v. During that time it waits to receive the votes of the rest of the alerted nodes and buffers them, as it has to wait for the key publishing phase in order to authenticate them.

The vote of each alerted node needs to reach all other alerted nodes. Since the messages are signed with a key known only to the sender, the attacker cannot change the votes. However, the attacker may refuse to forward votes, such that they must be forwarded through other paths, bypassing the attacker. Note that these paths can consist of more than two hops.

To ensure that the votes propagate to all alerted nodes, we follow a broadcast message-suppression protocol, similar to SPIN [6]. When an alerted node receives a vote, it advertises it, by broadcasting an ADV message. Upon receiving an ADV, each neighboring node checks to see whether it already has received or requested the advertised vote. If not, it sets a random timer T_{req} to expire, uniformly chosen from a predetermined interval. When the timer expires, the node sends a REQ message requesting the specific vote, unless it has overheard a similar REQ from another node. In the latter case, it cancels its own request, as it is redundant.

5.3   Publish Key Phase

In the Publish Key phase each node broadcasts the key of its hash chain, K_j, which was used to sign the vote. When a node receives the disclosed key, it can easily verify the correctness of the key by checking whether K_j generates the previous key through the application of F. If the key is correct, it replaces the old commitment K_{j-1} with the new one in its memory. The node now uses the key to verify the signature of the corresponding vote stored in its buffer from the previous phase. If this process is successful, it accepts the vote as authentic.

We allow sufficient time for the nodes to exchange their keys by setting a timer T_p. This timer is initialized just after a node publishes its own key and it is set to expire at time τ_p. During this time period, the nodes follow the same ADV-REQ scheme that we described above.

When the timer expires, the nodes move to the final step of processing the votes and exposing the attacker. In the case where a key has been missed, the corresponding vote is discarded.

Since nodes are not time synchronized, and some nodes may start publishing their keys while others are still in the voting phase, we need to consider "man in the middle" attacks. When a node sends its vote, an attacker may withhold it until that node publishes its key. Then it can change the vote, sign it again with the new key, and forward it to the next alerted node. Following that, the attacker also forwards the key, and the receiver will be able to verify the signature and accept the fake vote as authentic. We deal with this problem implicitly by relying on *residual paths* amongst the nodes. As votes are forwarded by all nodes, even if an attacker refuses to forward a vote, it will arrive to the intended recipients via other paths.

5.4   Exposing the Attacker

When each alerted node $s_1, s_2, \ldots, s_n$ has collected and authenticated the votes from all the other members of the alert region, it will have knowledge of all

the corresponding suspect lists, $D(s_1), D(s_2), \ldots, D(s_n)$, its own included. Then it applies a count operator which counts the number of times δ_i each node i appears in the suspect lists, in order to produce the final intrusion detection result, i.e., the attacker's ID. All alerted nodes will reach the same result, since they all apply the same operator on the same sets.

As we proved in Section 4.1, IDC is a sufficient condition for the intrusion detection problem. But if it does not hold, then NC needs to hold in order to successfully identify the attacker, as we proved in Section 4.2. Thus, if there is one node holding the majority of the nodes, we know that this is the attacker. If not, then the honest nodes which also collected the majority have non-alerted neighbors. So, the nodes move to a new phase, the external ring reinforcement phase, where these neighbors are called to support their honest neighbors. We describe this phase in Section 5.5, where we will see that in this case the attacker is revealed.

5.5   External Ring Reinforcement Phase

As we said, when there are other nodes that have the same set of alerted neighbors $\tilde{AN}$ with respect to the attacker, i.e., IDC does not hold, the voting process may be inconclusive, if these nodes collect the same number of votes. In this section, we present an algorithm where, if NC holds, the alerted region can distinguish amongst the prevailing candidates and find the actual one. So, for what follows we assume that NC holds, meaning that *all* neighbors of the attacker are alerted and that honest nodes collecting the majority of the votes have non-alerted neighbors.

Let us assume the set $\mathcal{P} = \{p_1, p_2, \ldots, p_k\}$ of the nodes collecting the same number of votes as the attacker, including the attacker itself. According to NC, the nodes in $\mathcal{P}$ do not have exactly the same neighborhood. Honest ones will also have other neighbors, which are not in alerted state and are going to play an important role in this phase. Therefore, we make the following definition:

Definition 4 (External Ring). *The external ring is defined as the set of nodes which are not members of the alerted region, but any of them is a direct neighbor of at least one alerted node.*

Figure 5 shows an example where nodes 96 (the attacker) and 76 have the same alerted neighborhood, and therefore collected the same number of votes during the voting phase, i.e., $\mathcal{P} = \{96, 76\}$. The circle in the figure shows the neighborhood of the attacker. The nodes in the external ring are represented by a triangle. The neighborhood of node 76 also includes the nodes 81 and 79, which are not alerted. These two nodes know that their neighbor 76 is not the attacker. If they share this information with the nodes in the alerted region, they can help them to distinguish the attacker.

The external ring reinforcement phase is initiated by the nodes in the alerted region. They broadcast a request to their non-alerted neighbors, including the set $\mathcal{P}$ in the message. The intended receivers check to see if any nodes in $\mathcal{P}$ are their neighbors and broadcast a message voting *in favor* of them.
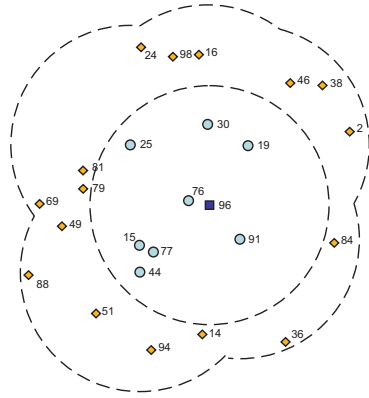
Fig. 5. The attacker's external ring is defined by the nodes which are 2-hops away from the attacker.
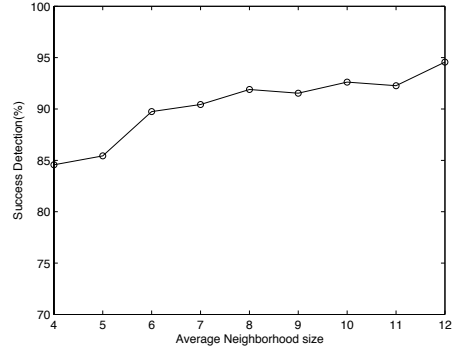

Fig. 6. The overall success rate of the simulated intrusion detection protocol for different neighborhood sizes.

The response message sent by any node of the external ring is forwarded by alerted nodes as in previous phases, such that it can reach all nodes in the alerted region. It is also signed using the next key in the key chain of the sender. The key is released after some fixed period of time and used by the receivers to authenticate the message.

6   Simulation Results

We simulated a sensor network of 100 nodes placed uniformly at random in order to test our intrusion detection algorithm. Figure 6 shows the probability that the IDS system successfully identifies the attacker. To calculate it we run the simulation in $10,000$ different topologies, choosing each time a random attacker. If the voting phase was conclusive the protocol ended, otherwise the external ring reinforcement phase was activated. As the subsequent analysis showed, the cases where the protocol did not succeed were all due to the fact that IDC or NC were not satisfied in the given situation.

7   Implementation

In this section, we present experimental results from our implementation of the proposed IDS algorithm. The goal is to exhibit a reference implementation and check the feasibility of an IDS without focusing on its efficiency. Even so, the result shows that such a system is lightweight enough to be a viable and realistic solution from the real deployment perspective. Moreover, various efficiency improvements of the implementation are possible. We leave them to future work.

7.1   Memory and Computational Requirements

The memory footprint of the our implementation is an important measure of its feasibility and usefulness on memory constrained sensor nodes. Table 1 lists the memory footprint of the modules, compiled for the MSP430 microcontroller.

Table 1. Size of the compiled code, in bytes.

Module	RAM	Code Size
Neighborhood Discovery	136	968
Exchange of Keys	184	3628
Reliability (ADV-REQ)	104	32
Voting	159	4844
Total	583	9472

The largest module in terms of RAM footprint is the Key Management module. Its size depends on the maximal number of node's neighbors which is configurable and currently set to 8. For each neighbor, a 10-byte key must also be stored. In terms of ROM, the largest module is the Voting module, since it has the most lines of code. In total, the IDS leaves enough space in the mote's memory for user applications. For example, Tmote Sky has 10 KB of RAM and 48 KB of program memory.

7.2   Experiments

To evaluate the performance of the implementation of the IDS, we tested it in a real environment. We deployed several nodes in random topologies on the floor of an office building, set a node to be the "attacker" and gradually incremented the number of its neighbors to form larger alert regions. For each alert region size, we repeated the experiment for 20 random topologies.

The experiments were performed with motes running a typical monitoring application. We loaded the Delta application, where the motes report environmental measurements to the base station every 5 seconds. We also deployed the MultihopLQI protocol at the routing layer, which is an updated version of the MintRoute protocol [12] for the Chipcon CC2420 radio. We tuned it to send control packets every 5 seconds. Our goal was to investigate how well the IDS would perform under the presence of traffic on other layers. Then we started an attack to trigger the IDS protocol.

Figure 7(a) depicts the communication cost of the protocol measured in packets sent by a node. In particular, we broke it down to the packets exchanged for the voting phase and the publish key phase (as a total of exchanging the votes, ADV, REQ and keys). As expected, the number of packets exchanged in the two phases is almost the same, since the message dissemination protocol does not change. For small alert region sizes the cost is only about 12 packets, while
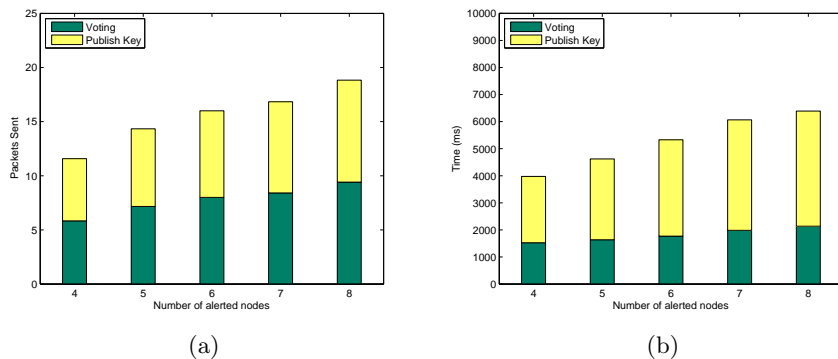
Fig. 7. (a) Measured communication cost for different number of alerted nodes. (b) Detection time for different number of alerted nodes.

for more dense regions the cost still remains low (19 packets). This is the total communication cost per attack and involves only the nodes in the alert region. The number of packets depends on the topology and the number of the alerted nodes, which determine the number of votes and keys circulated amongst them.

Next we measured the time that each phase of the IDS protocol required, i.e., the voting phase and the publish key phase. Figure 7(b) shows the measured mean times for each of the above phases, for different number of alerted nodes (i.e., attacker's neighborhood). We can see that the time for the voting phase increases only moderately as the number of alerted nodes increases, and contributes the smallest overhead in the total delay. The most time-consuming phase is the publish key phase, where nodes exchange their keys and verify the votes. To get a better insight, we measured the time needed for the computational operations within this phase. It turns out that the computations (key verification, validation of the signatures on the votes, and the construction of the final result) are responsible only for about 30% of the consumed time, whereas the communication is responsible for the rest.

8   Conclusions and Future Work

In this paper we made a first attempt to formalize the problem of intrusion detection in sensor networks, and showed the benefits and theoretical limitations of the cooperative approach to intrusion detection. We presented necessary and sufficient conditions for successfully exposing the attacker and a corresponding algorithm that is shown to work under a general threat model.

Our investigation of the case of a single attacker ($t = 1$) gave very valuable insights into the solvability of cooperative intrusion detection. In future work we plan to concentrate on the case where the attacker can capture more nodes ($t > 1$). We also plan to look into dynamic neighborhood changes, in particular, into secure node addition and removal in sensor networks.

The intrusion detection algorithm we discussed is the first *generic* algorithm for intrusion detection in sensor networks. Although individual solutions to specific problems might be more efficient, our reference implementation demonstrates that our algorithm is lightweight enough to run on sensor nodes. Thus, studying the problem of intrusion detection in sensor networks is a viable research direction and with further investigation it can provide even more attractive solutions for securing such types of networks.

References

1. V. Bhuse and A. Gupta. Anomaly intrusion detection in wireless sensor networks. *Journal of High Speed Networks*, 15(1):33–51, 2006.
2. A. P. da Silva, M. Martins, B. Rocha, A. Loureiro, L. Ruiz, and H. C. Wong. Decentralized intrusion detection in wireless sensor networks. In *Proceedings of the 1st ACM international workshop on Quality of service & security in wireless and mobile networks (Q2SWinet '05)*, pages 16–23. ACM Press, October 2005.
3. F. Kargl, A. Klenk, M. Weber, and S. Schlott. Sensors for detection of misbehaving nodes in MANETs. In U. Flegel and M. Meier, editors, *Detection of Intrusions and Malware & Vulnerability Assessment, GI SIG SIDAR Workshop, DIMVA 2004, Dortmund, Germany*, volume 46 of *LNI*, pages 83–97. GI, 2004.
4. I. Krontiris, T. Dimitriou, and F. C. Freiling. Towards intrusion detection in wireless sensor networks. In *Proceedings of the 13th European Wireless Conference*, Paris, France, April 2007.
5. I. Krontiris, T. Dimitriou, T. Giannetsos, and M. Mpasoukos. Intrusion detection of sinkhole attacks in wireless sensor networks. In *Proceedings of the 3rd International Workshop on Algorithmic Aspects of Wireless Sensor Networks (AlgoSensors 07)*, Wroclaw, Poland, July 2007.
6. J. Kulik, W. Heinzelman, and H. Balakrishnan. Negotiation-based protocols for disseminating information in wireless sensor networks. *Wireless Networks*, 8(2/3):169–185, 2002.
7. L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.
8. C. E. Loo, M. Y. Ng, C. Leckie, and M. Palaniswami. Intrusion detection for routing attacks in sensor networks. *International Journal of Distributed Sensor Networks*, 2005.
9. A. Mishra, K. Nadkarni, and A. Patcha. Intrusion detection in wireless ad hoc networks. *IEEE Wireless Communications*, 11(1):48–60, February 2004.
10. I. Onat and A. Miri. An intrusion detection system for wireless sensor networks. In *Proceeding of the IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, volume 3, pages 253–259, August 2005.
11. M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.
12. A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 14–27, 2003.