TinyAFD: Attack and Fault Detection Framework for Wireless Sensor Networks

Răzvan Rughiniș, Laura Gheorghe

Faculty of Automatic Control and Computers, University Politehnica of Bucharest, Bucharest, Romania.
(e-mail: {razvan.rughinis, laura.gheorghe}@cs.pub.ro)

Abstract: Wireless Sensor Networks (WSNs) are especially vulnerable to malicious attacks because of the limited resources available on the sensor nodes. When conducted against WSNs that run critical applications, such as military surveillance, these attacks may threaten human lives. Therefore, it is important to integrate strong security mechanisms for protecting critical applications that run on top of WSNs. In this paper, we propose a security solution for detecting attacks and faults in WSNs, called Tiny Attack and Fault Detection Framework (TinyAFD). The framework is modular, as it consists of engines able to detect different attacks based on signatures, providing the possibility to modify signatures at runtime and to easily integrate new specialized engines. We implemented TinyAFD in TinyOS and conducted multiple tests in order to evaluate the functionality and performance of the framework. In comparison with alternative solutions, TinyAFD is modular, configurable and extensible, and it mitigates most well-known attacks, while being energy efficient.

Keywords: Wireless Sensor Network; Security; Signature-based detection; TinyOS; Energy efficiency.

1. INTRODUCTION

A Wireless Sensor Network (WSN) consists of a large number of low-cost, low-power, resource-constrained devices that collect data from the environment. They collaborate with each other in order to transfer information to a centralized location where it is further processed and analyzed (Zheng et al. 2009).

Due to the resource-constrains of sensor nodes and the wireless medium used for communication, WSNs are vulnerable to malicious attacks. Potential threats include injecting malicious packets into the network to influence event detection or routing, or generating a large number of packets in a small period of time in order to consume the energy of sensor nodes (Zia and Zomaya 2006; Bojkovic et al. 2008).

WSNs may be used in critical applications, such as military surveillance, medical monitoring, and homeland security. When malicious attacks are conducted against critical applications, they may influence the results produced by the sensor network and may cause the loss of resources or even human lives. Therefore, the critical applications must be protected against malicious attacks using strong security solutions.

While computer networks are protected using firewalls and intrusion prevention systems, these solutions have to be adapted or re-designed in order to serve the needs of low-power equipment such as sensors nodes (Vasseur 2011). The security mechanisms for WSNs should be designed with resource efficiency in mind (Voinescu et al. 2010).

An intrusion detection system should be modular, configurable and extensible because each application that runs on top of a WSN has its own security requirements and it is exposed to specific threats. In addition, the intrusion detection system should cover most attacks against WSNs. However, the available solutions for WSNs do not satisfy these requirements. There is a need for developing a modular framework that can be easily reconfigured or extended.

In this paper, a new intrusion detection system for Wireless Sensor Networks, called Tiny Attack and Fault Detection framework (TinyAFD) is proposed. TinyAFD has modular design and it consists of several detection engines bound together in a detection chain: MAC engine, Atomic engine, Flood engine, Time Correlation engine and Space Correlation engine. These engines monitor the traffic routed by the current node. TinyAFD also includes the Routing engine, which is not part of the chain, but intercepts all packets received and sent by neighbor nodes in order to analyze their routing behavior. This way, TinyAFD allows for the detection of a large number of attacks generated by internal faulty and malicious nodes. The engines use fine-tuned signatures in order to identify intrusion attempts and mitigate them.

The rest of this paper is structured as follows. Section 2 discusses the problem, the state-of-the-art intrusion detection solutions for sensor networks and the requirements for improving intrusion detection mechanisms. Section 3 describes the solution for attack and fault detection in WSNs. Section 4 details the implementation in TinyOS. Section 5 presents the experimental methodology and results. Section 7 concludes the paper and proposes directions for future research.

2. RELATED WORK

In this section, we are going to present the problem that has been tackled in this paper, the current solutions to this

problem and the requirements for improving intrusion detection mechanisms.

2.1. The Problem

A large number of malicious attacks can be conducted against WSNs. The attacker may take advantage of the wireless medium by injecting malicious packets into the network. Also, he may profit from the fact that sensor nodes have limited resources and deploy Denial of Service attacks against them. The purpose of the attacker may be to steal critical data or to influence the results produced by the WSN.

The attacks that can be conducted against WSNs have been analyzed by (Wang et al. 2006): packet injection and alteration (including the Sybil attack), replay (including the Wormhole attacks), Selective Forwarding, Sinkhole, Blackhole, de-synchronization, and flood attacks. These attacks take place at the network and transport layer. Other important attacks may take place at link and physical layers: jamming, tampering, eavesdropping, node capturing, collision, exhaustion, and unfairness.

When deployed against critical applications, these attacks may cause damage to resources or people. Critical applications that run on top of WSNs should be protected against malicious attacks. For that, attacks must be identified and blocked using intrusion detection and prevention systems.

2.2. Current Solutions

Intrusion Detection and Prevention Systems (IDPS) are complex software applications or hardware devices that are able to identify intrusion attempts, block them, and send alerts to a centralized location where they can be observed by system administrators.

Intrusion detection solutions are usually classified into signature-based or anomaly-based systems. Anomaly-based solutions compare network traffic with normal traffic profiles that have been built during an initial learning phase. This method is efficient for unknown threats, but it has the disadvantage that threats cannot be unambiguously identified and differentiated from false positives (Scarfone and Mell 2007).

Signature-based solutions work with a pre-defined set of threat descriptions, called signatures. The network traffic is constantly compared with signatures, aiming to find a match and thus to identify that specific threat. The disadvantage of this method is that it cannot detect unknown threats, while its advantages consist of low rates of false positives and a relatively straightforward operation.

Signature-based solutions may offer considerable advantages of accuracy, energy saving, and low false positive errors. Significant proposals have been advanced by (Silva et al. 2005; Strikos 2007; Yan et al. 2009; Hai et al. 2010; Stetsko et al. 2010).

(Silva et al. 2005) present a rule-based decentralized intrusion detection system for sensor networks. The monitor nodes detect events and analyze network behavior. The IDS is adapted to a specific network in several steps: (1) select rules used to monitor specific features defined by the user; (2) compare the information that can be gathered by the network with the information found in the rules in order to select the final set of rules; (3) configure the parameters of the final rules with their values from the network design definitions.

(Strikos 2007) reports stepwise the construction and deployment of an intrusion detection system for sensor networks with hierarchical design. A “cluster first” protocol is used: clusters are built, and, subsequently, cluster heads are chosen. The IDS is placed on cluster heads, and it monitors packets with reference to the rule set specified by (Silva et al. 2005). Rule violation may trigger an alarm and, if the alarms reach a specified threshold for a node, it is diagnosed as an intruder and it is removed from the network.

(Yan et al. 2009) propose a Hybrid Intrusion Detection System for Cluster-based Wireless Sensor Networks (CWSN) that aims to prolong the life of the sensor nodes. It uses a hybrid detection model in order to combine the high detection rate of anomaly detection with the accuracy of misuse detection. Decisions are taken by a specialized decision-making model based on the outputs of anomaly detection and misuse detection techniques. The decision making model also classifies attacks and notifies the administrator.

(Hai et al. 2010) propose a lightweight intrusion detection framework for Cluster-based Sensor Networks, implemented at the application layer. The framework consists of a local agent and a global agent. The local agent monitors the information sent or received by the current node, and it also gradually builds and stores a blacklist with information about malicious nodes. The node uses this blacklist as a signature database, and signatures are propagated to other nodes. The global agent overhears the communication of its neighbors and uses a watchdog monitoring mechanism to detect anomalies in neighbors’ routing processes.

(Stetsko et al. 2010) is a collaborative intrusion detection system, in which neighbor nodes communicate with each other in order to detect Selective Forwarding, Hello Flood and Jamming attacks. The collaboration between neighbor nodes reduces the false negatives rate. The mechanism has been chosen for the implementation of Collaboration Tree Protocol (CTP).

2.3 Required Improvements

(Rassam et al. 2012) present a comprehensive survey of the current intrusion detection systems for Wireless Sensor Networks. They enumerate some requirements that must be satisfied in order to improve the current solutions.

One important requirement is the *detection generality*, which means that an innovative solution would be able to detect a large range of attacks. In this paper, we propose a solution that is able to detect most well known attacks and that can be easily extended to detect new attacks. It is very important to cover as many attacks as possible using a single security solution.

Another requirement is the *detection speed*, which means that the detection of attacks must take place as soon as possible in order to be efficient. The proposed framework does not involve collaboration between sensor nodes and takes the decisions on the detector nodes based on the analyzed traffic, which improves the detection speed considerably.

Another important requirement is the *distribution of intrusion detection*, which refers to the fact that the IDS agent should run on multiple nodes in the network in order to increase detection efficiency. The system proposed in this paper runs on multiple nodes called detector nodes.

3. ATTACK AND FAULT DETECTION FRAMEWORK FOR WIRELESS SENSOR NETWORKS

Wireless Sensor Networks that run critical applications should be protected against attacks and faults.

In this paper, we propose a framework for detecting attacks and faults in Wireless Sensor Networks, called Tiny Attack and Fault Detection framework (TinyAFD).

TinyAFD is based on the architectural principles proposed in the signature-based Intrusion Detection and Prevention System (IDPS) for computer networks implemented in the Linux kernel, the Attack Evaluation and Mitigation Framework (Gheorghe et. al. 2010a).

3.1. Notations and Assumptions

We assume that a WSN is a collection of devices, each having one or more sensors that can collect data from the environment. A WSN can be represented using a graph, as in Formula 1, where *V* is the collection of sensor nodes and *E* is the collection of communication links between the nodes.

$$WSN = (V, E) \tag{1}$$

The collection of nodes can be represented as in Formula 2, where N_i is a sensor node.

$$V = \bigcup N_i \tag{2}$$

A link between two nodes can be represented as in Formula 3, where N_i and N_j are two sensor nodes that can communicate with each other.

$$E = \{N_i, N_j\} \tag{3}$$

The WSN may have a single-hop or multi-hop logical topology. The multi-hop topologies are divided into flat and hierarchical topologies. The hierarchical topologies can be classified into single-hop or multi-hop cluster topologies and into single-tier and multi-tier cluster topologies. The routing protocol dictates the logical topology.

In regard to the attacker model proposed by (Wang et al. 2006), it is assumed that the attacker may be an insider or outsider, active or passive, able to generate both mote-class or laptop-class attacks. While considering all cases described by Wang, there is a main focus on insider compromised nodes that generate mote-class, active attacks.

In relation to the attacker model proposed by (Benenson et al. 2010), attacks against all security requirements are addressed. Also, it is assumed that the attackers may be both clever outsiders and knowledgeable insiders.

Regarding the type of presence, it is assumed that the attackers may be either local or distributed, deploying one or more sensor nodes in the network and coordinating them.

As regards malicious interventions, disturbing attackers are assumed. Accordingly, attackers may try to perform eavesdropping, inject malicious packets, alter packets on their way to destination, and disrupt the network functionality.

Using the model of (Benenson et al. 2010), an attacker can be represented as A(presence,intervention). An attacker A(distributed,disturbing) is considered.

Most types of attacks should be considered when designing an intrusion detection solution, in order to provide detection generality.

It is assumed that network initialization includes pair-wise key deployment. When new nodes are added to the network, they should authenticate themselves before being included in the communication. Therefore, an internal node may have a malicious behavior only if the node is physically captured and manipulated by the attacker. Also, an internal node may act abnormally in case of software or hardware failures.

It is assumed that the network is dynamic: a new node may join the network or a node can leave the network. The proposed framework must be resilient to the changes in network topology.

3.2. Framework Design

TinyAFD has a modular design, being composed of building blocks called engines, which use signatures to identify abnormal behavior.

Engines are software modules that interpret a specific type of signature. The default engines in TinyAFD are the MAC, Atomic, Flood, Time Correlation, Space Correlation and Routing engines. However, a developer may easily extend the framework by implementing new engines according to the requirements of a specific context.

A modular approach provides considerable flexibility. Except for the Routing engine, the engines are placed in a chain, as represented in Figure 1, from lowest to the highest computational complexity and storage requirements.

The packets that are not blocked by a certain engine are sent to the next engine. If the packet is not blocked by the last engine in the chain, it is accepted by TinyAFD and sent to the next layer. If the packet is blocked by any of the engines, the packet is dropped by TinyAFD, and it is not sent to the next layer.

The MAC, Atomic, Flood and Correlation engines analyze each packet received by the node. They are placed in the communication stack between the Link Layer and the

Network Layer. Each packet routed by, destined to, or sent by the current node is analyzed by these engines.

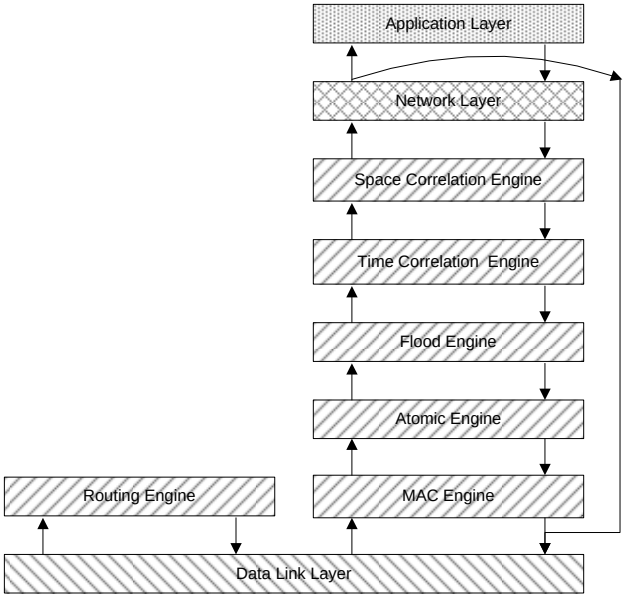


Fig. 1. TinyAFD Architecture.

The Routing engine intercepts the packets received and sent by the neighbor nodes in order to detect malicious behavior. The Routing engine is placed on top of the Data Link layer.

3.3. Detector Nodes

In order to minimize energy consumption, TinyAFD runs only on selected nodes called detectors, which should be placed in key positions in the sensor network.

For example, in a hierarchical network, cluster heads should be detectors. However, the TinyAFD is present on all sensor nodes and it may be enabled and disabled according to situational demands. For example, when the cluster head is re-elected, the new cluster head enables TinyAFD while the old cluster head disables it.

3.4. Signatures

TinyAFD is a signature-based system aiming at maximizing resource efficiency while covering a wide range of attacks. Each signature can be fine-tuned for specific networks, protocols and applications.

Each signature is defined as a symptom-treatment pair, including a set of conditions that define an intrusion, on the one hand, and an action to be taken when a packet or a sequence of packets match the conditions, on the other hand.

Signatures may be classified, according to the amount of information they require, into atomic signatures and signatures that require a comparative evaluation of several pieces of information.

Additional signatures can be easily added to the framework whenever a new threat is identified. The ease in adding new types of signatures is a significant advantage of the TinyAFD framework.

3.5. Actions

TinyAFD may decide, on the basis of the available signatures, whether to block a suspicious packet. TinyAFD is able to block all packets coming from a certain source node, all packets in a conversation between a pair of source and destination nodes, or just a single packet.

TinyAFD decides, according to signature instructions, whether to announce the administrator about the intrusion attempt or not. This possibility allows for energy efficiency optimization by decreasing the number of alert packets.

An advantage of TinyAFD in the context of Wireless Sensor Networks is that malicious packets may be dropped at the first detector node. Therefore, energy consumption damage is minimized, and the malicious effects upon the routing and application layers are controlled.

3.6. Engines

Having a modular design, TinyAFD includes 6 default engines: MAC, Atomic, Flood, Time Correlation, Space Correlation and Routing engine.

3.6.1. MAC Engine

The MAC engine is the first engine in the chain to analyze received packets. The MAC engine is responsible with authenticating received packets and uses a Message Authentication Code (MAC) method.

The engine validates authentication and integrity; therefore, it covers multiple attack cases: packets generated by external nodes, packets altered by external and by communication faults. The MAC engine protects the network against packet injection and packet altering.

3.6.2. Atomic Engine

The second engine in the chain is the Atomic engine, which checks the validity of header fields and verifies the correctness of the payload. The packets that reach this engine are generated by internal nodes, because they have been validated by the MAC engine.

The engine does not store any packet information. The signatures represent general rules regarding the header fields and payload. These rules specify ranges for the inspected values.

The rules for the header fields and payload are determined from the communication protocol and application specification.

3.6.3. Flood Engine

For any conversation under analysis, the Flood engine stores and modifies three parameters for each received packet: rate, peak and gap (Rughiniș and Gheorghe 2010).

The Flood engine also specifies three limits associated with parameters. The rate limit indicates the number of packets per second that indicates suspicious traffic. The peak limit

defines the period of time during which the rate of packets should be greater than the limit specified in the rate field, required to match the signature. The gap limit indicates the period of time with traffic below the specified rate required to reset the peak count to 0.

If the malicious node generates packets with a rate equal or greater than the rate value included in the signature for more that peak seconds, the signature is matched and the subsequent packets from that source node are dropped.

When a packet is received, the rate is incremented. After each second, the rate is verified, and, if it is less than the rate from the signature, the gap is incremented. When the gap exceeds the limit, the peak is set to zero. If the rate is greater than the limit, the peak is incremented and the gap is set to 0. After that, the rate is set to zero. If the peak reaches the limit, the alert is triggered.

A firewall module can be integrated with TinyAFD in order to block subsequent packets based on rules generated by the engines.

3.6.4 Time Correlation Engine

The Time Correlation algorithm is based on the principle that, given a sufficiently high frequency of measurement, sensors cannot generate two values, which are very different, in a short period of time. When this happens, the source node is either faulty or compromised.

The Time Correlation engine stores a set of packets from each conversation in order to compare the sensed data. A timer is used to verify that the time difference between the analyzed packets is less than the pre-defined one (time limit). If the time difference is greater, the analysis cannot be performed, because over larger periods of time the sensed values may change significantly, especially in the case of a critical event. The time limit is used in order to prevent false positives caused by packet loss.

The engine determines the median value for the stored packets and compares it with the one found in the received packet. If the difference is greater than the specified threshold value, the alert is triggered.

For error detection purposes, the median is a better measure of the central tendency than the average, because it is less sensitive to outliers.

3.6.5. Space Correlation Engine

The Space Correlation engine stores a set of packets for each group of nodes, which are in close proximity with each other.

The principle behind the Space Correlation operations is that sensor nodes that are in the proximity should have similar collected data. However, if a node is malicious or faulty, it can generate data sets that do not match the data generate by its neighbors. Therefore, an incorrect data reading can be detected by comparing the data collected by sensor nodes from the same area.

A geographical clustering method is used to identify groups of nodes, such as the one specified in (Gheorghe et al. 2010b). The validation method is shared with the time correlation engine: for each group of nodes each new packet is evaluated against the median value of the stored ones.

3.6.6. Routing Engine

The Routing engine monitors neighbor nodes in order to determine routing abnormalities. The engine puts the node in promiscuous mode and listens to network traffic, which is received and sent by neighbor nodes.

A node that has normal behavior forwards the exact packet which it has received. If the inspected node does not forward the packet or alters the packet (except for link layer headers) it is considered suspicious.

3.7. Reconfiguration Commands

The default signatures are stored on the sensor nodes from deployment. If the network user wants to alter, modify or delete a signature, it sends a command packet to the base station, which disseminates the packet in the entire network. A simple dissemination protocol such as Sensor Protocols for Information via Negotiation (SPIN) can be used (Heinzelman et al. 1999).

The modify command is used for altering an existing command. The add command is used for adding a new signature or for completely replacing an old one. The delete command is used in order to delete an existing signature from TinyAFD.

Commands are disseminated in the whole network in order to provide synchronization between current detector nodes and future ones. Although it is an expensive operation, it is performed less often than data delivery, so it consumes less energy than normal data delivery operation.

Command packets are encrypted using a block cipher and authenticated using a Message Authentication Code (MAC).

3.8. Resiliency to Topology Changes

Topology change is an important consideration for any monitoring and evaluation solution (Bardac et al. 2009). Three cases, in which the network topology may change have been encountered during experimental evaluation: a node joins the network, a node leaves the network and a node changes its position due to mobility.

When a new node joins the network, the closest detector node is able to analyze the traffic generated by or the routing behavior of the new node.

When a node leaves the network due to battery depletion or mobility, some problems may arise. The node may have received some packets, and then died without sending them to the next hop. In addition, a node may die after sending some faulty data. An alert may be generated regarding a dead node. However, the administrator observes that the dead node is not sending data anymore, so he will not consider the alert.

When a node changes its position due to mobility, it may cause some problems. A node may receive some packets to forward, being observed by a detector node, then changed position, and sent the packets to the next hop, being observed by another detector. The first detector generates an alert regarding malicious routing behavior. The administrator receives information regarding the position of the nodes, he observes that the position of the node has changed and does not consider the alert. This can be considered a small price for mobility.

There are no such problems regarding external attackers, which are detected using the authentication mechanism. In this case, it is irrelevant if the malicious node dies or changes position.

4. IMPLEMENTATION

4.1. TinyOS and Communication System Hooks

TinyOS is an open-source operating system that has been designed to run on low-power devices, including sensor nodes (Levis et al. 2004).

Communication System Hooks are implemented using components linked in the TinyOS kernel in order to be able to analyze all traffic that the sensor node receives or sends. In order to receive all packets, one HookReceiverP component is linked with the ActiveMessageImplP component and with the AMReceiverC, using a wiring component HookReceiverC. Likewise, in order to analyze and modify sent packets, the HookSenderP component is linked with the AMSenderC, AMQueueEntryP and ActiveMessageC using the wiring component HookSenderC.

4.2. TinyAFD Implementation

TinyAFD has been implemented as a new layer in the communication stack of the TinyOS kernel, using the Communication System Hooks method. The component-oriented nature of TinyOS permits a high level of modularity. Each TinyAFD engine has been implemented in a TinyOS

component in the kernel, and the engines are wired together in a chain. Each engine is implemented using two Hook components: one for the actual implementation, and one for the wiring.

The MAC engine is the first filter in the concatenation, because it checks packet integrity and authentication. This engine is able to compute a MAC using the parameters specified in the signature and to check it against the one found in the received packet. HMAC implementation has been chosen for the test scenarios, because it is accepted as an effective solution for low-power devices (Arazi 2009; Lee, 2010).

The second engine in the chain is the Atomic engine, which checks the validity of header fields and payload. The component is easy to implement because it does not store any packet information, unlike the Flood or Correlation engines.

The Flood engine is the third engine in the chain. For any conversation under analysis, TinyAFD stores and modifies three parameters for each received packet: rate, peak and gap. A MilliTimer is used to verify the traffic state each second, in order to check if they match the flood signature.

The Time Correlation engine stores a set of packets from each conversation as a comparison benchmark. A MilliTimer is used to verify that the time difference between the analyzed packets is less than the one found in the signature.

The Space Correlation engine also stores a set of packets for each territorial group of nodes. A MilliTimer is used in the validation algorithm.

The Routing engine intercepts all packets that are not destined to the local node and that are either sent, or received by neighbor nodes. The engine stores these packets and analyzes them with reference to existing signatures, in order to detect malicious routing behavior.

Several actions can be implemented as a response to signature triggering. The most common decision after detecting an intrusion is to block the malicious packet. Another common action is to send an alert to the network administrator via the base station.

Another action may consist in blocking all packets coming from a certain node. This requires the implementation of a mini-firewall linked with TinyAFD. This firewall is placed on top of the MAC engine that checks whether the source address has not been spoofed. This is important in order to avoid punishing sensor nodes whose identity has been spoofed by the malicious node.

5. EVALUATION METHODOLOGY AND EXPERIMENTAL RESULTS

The efficiency of TinyAFD has been evaluated, by running a set of test scenarios using the TOSSIM simulator for TinyOS applications (Levis et al. 2003).

5.1. Functionality evaluation

The test infrastructure consists of a sensor network with 20 nodes, as shown in Figure 2. We consider a real-world test case, in which the nodes are used for monitoring a datacenter room, they gather information from a temperature sensor and send it towards the base station for further analysis.

A cluster-based routing protocol is used, so the cluster heads to act as detector nodes. In this scenario, it is assumed that the cluster heads that have been elected are nodes 5, 12, and 20, while node 1 is the Base Station.

Each cluster head analyzes only traffic coming from its own cluster and knows exactly its membership. In the experimental setup there are three clusters: (2, 3, 4, 5, 6, 7), (8, 9, 10, 11, 12, 13, 14), and (15, 16, 17, 18, 19, 20).

In the test scenarios the attacker is either an external device introduced into the monitored datacenter room, that does not have knowledge about the cryptographic keys needed for communicating in the network, or a internal sensor node that

has the cryptographic keys and is compromised by an external attacker. Multiple attacks have been conducted, in order to evaluate the detection ability of TinyAFD.

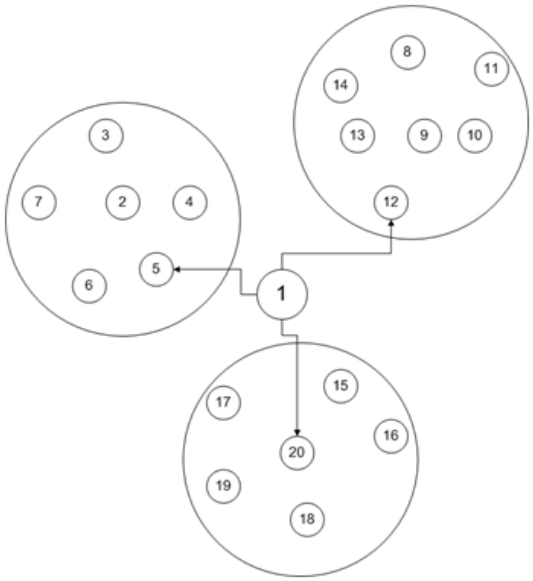


Fig. 2. Toplogy for Testing TinyAFD.

5.1.2. Invalid Data Injection Attack

Invalid data injection may be performed by external attackers or by internal compromised or faulty nodes.

5.1.2.1. Invalid Data Injection Performed by External Attacker

In this scenario, an external malicious node poses as a valid node within a cluster and sends a packet with correct sensed data. However, because it is an external node, it does not have the secret key to compute a correct MAC.

The MAC engine has the following signature: *(HMAC, payload, seqno, secret, drop).* The MAC engine computes the MAC using the packet payload, the sequence number found in the packet, and the shared key that is known by the detector node. The computed MAC does not match the one found in the packet; therefore, the packet is dropped as specified in the signature.

5.1.2.2. Detection of Invalid Data Injection using the Time Correlation Engine

In this scenario, an internal node is compromised or faulty and begins generating invalid data packets. The attack is detected using the Time Correlation engine.

In the Time Correlation Application engine a signature like *(SenseID, MaxValue, MaxTime, MaxPackNo, Action)* can be used to block any packet that contains a sensed value that is significantly different than the last values provided by that sensor node. This type of signature is useful for preventing compromised notes from sending invalid packets.

In the attack simulation, node 7 sends packets with values 20, 19, 21, and then it is compromised and sends a packet with invalid data 27. The correlation signature is *(temperature, 2, 1000, 3, drop).* The invalid packet sent by node 7 is blocked by TinyAFD.

5.1.2.3. Detection of Invalid Data Injection using the Space Correlation Engine

In this scenario, an internal compromised or faulty node that generates invalid data packets is considered.

The Space Correlation engine has the following signature: *(SenseID, MaxDiff, MaxDist, MaxTime, MaxPackNo, Action).* In this case, the engine stores a specified maximum number of packets that are coming from the same group of nodes and blocks any packet that contains data that is significantly different from data generated by other nodes in the same group.

The correlation signature is *(temperature, 3, 1000, 1000, 5, drop).* In the scenario, node 7 has been compromised and it sends invalid data value 29. As observed in Listing 1, the packet is blocked due to space correlation mismatch.

```
(2): ApplicationLayer: Packet sent [src=2 dest=1 seq=23 data=14]
(4): ApplicationLayer: Packet sent [src=4 dest=1 seq=23 data=15]
(3): ApplicationLayer: Packet sent [src=3 dest=1 seq=23 data=12]
(6): ApplicationLayer: Packet sent [src=6 dest=1 seq=23 data=13]
(5): ApplicationLayer: Packet sent [src=5 dest=1 seq=23 data=16]
(7): ApplicationLayer: Packet sent [src=7 dest=1 seq=23 data=29]
(5): TinyAFD: Space Correlation engine: diff=15. Packet is blocked.
[src=7 dest=1 seq=23 data=29]
```

Listing 1. Attack Blocked by the Space Correlation Engine

5.1.2.4. Detection of Invalid Data Injection using the Atomic Engine

If the attacker sends values outside the normal range, its packets can be blocked using the Atomic engine with a simple signature such as *(SenseID, MinValue, MaxValue, Action).* Parameters indicate that, if the value contained in the packet is less than the *MinValue* or greater than *MaxValue*, the packet is dropped.

In the attack simulation, the malicious or faulty node sends an invalid data value. The Atomic engine uses the temperature signature to check the validity of the packet and, after it detects an abnormal value, it drops the packet.

5.1.3. Flooding Attack

In this scenario, the malicious node generates and sends a large amount of packets that have the same source address. The Flood engine includes signatures such as: *(Type, Rate, Peak, Gap, Action).*

In the attack simulation, in Listing 2 the malicious node 7 generates more than 100 packets per second. The signature used to detect this attack is *(source_based, 100, 2, 1, drop).* The malicious node generates more than 100 packets during two seconds (the first and the third) and less than 100 packets during one second. The attack is detected after 3 seconds, and the cluster head can block the subsequent packets that come from node 7.

```
(7): [247] ApplicationLayer: Packet sent [src=7 dest=1 seq=19 data=16]
[..]
(7): [3099] ApplicationLayer: Packet sent [src=7 dest=1 seq=250
data=19]
(5): [3001] TinyAFD: Flood engine: Flood attack detected. Rate=1
Peak=2 Gap=0 [src=7 dest=1 seq=250 data=19]
```

Listing 2. Detection of a Flood Attack

5.1.4. Blackhole Attack

In this scenario, a neighbor node has a Blackhole behavior: it does not forward to the destination any of the received packets. The Routing engines include signatures such as: *(BehaviorID, Lim, Action)*. In this case, the signature: *(blackhole, 10, alert)* is considered - when at least 10 packets are dropped by the suspicious node, the signature is triggered and an alert is sent to the base station.

In the simulation, the cluster head 12 suddenly begins to drop packets received from the cluster members. When 10 packets are not forwarded by node 12, the signature is triggered and the base station is notified.

5.1.5. Selective Forwarding Attack

In this attack, a neighbor node selectively forwards packet towards the destination. The following signature is triggered: *(selective, 10, alert)*: more than 10 packets from a total of 20 are dropped by a node.

In the simulation, cluster head 12 forwards only 9 from 20 packets. The engine intercepts 20 unique packets that are handled by cluster head 12 (forwarded or not). If more than 10 packets are not forwarded, the signature is triggered and the Base Station is notified.

5.1.6. Packet Altering Attack

In this scenario, a cluster head modifies the data packets received from neighbors before forwarding them to the Base Station. The Routing engine includes the following signature: *(altering, 5, alert)*.

The cluster head 12 modifies at least 5 consecutive packets received from the cluster members. The engine intercepts the packets received and sent by node 12 and compares them. It decides that 5 packets have been altered by node 12, so the signature is triggered and the Base Station is notified.

5.2. Performance Evaluation

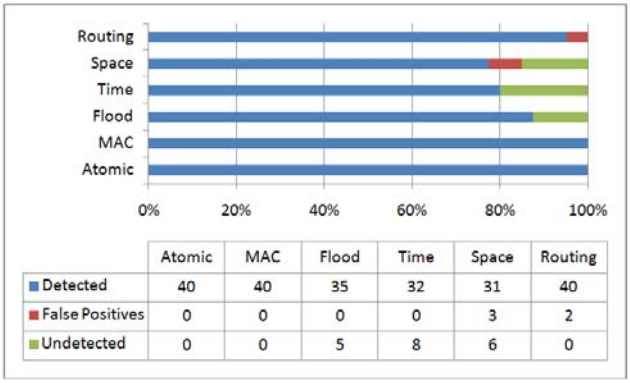
For evaluating the detection efficiency and resource efficiency, the detection rate, false positives, detection time, energy consumption and memory occupation for each detection engine have been determined.

5.2.1. Detection Rate and False Positives

An attack scenario for each engine has been designed, in order to determine the detection rate and false positives. The modeled attenuation and interferences produce a different rate of packet loss for each program execution; therefore, different results for different executions of the same scenario can be expected. For this reason, each attack scenario has been run 40 times, and the number of detected attacks, false positives, undetected attacks and average detection time have been detected. Each scenario ran during the same period of time.

For the MAC, Atomic, Flood, Space and Time Correlation engines, it is considered that the attacker is 2 hops away from the detector node. Therefore, packet loss can occur on any of the 3 communication links between the communicating nodes. The number of detected attacks and detection time are both influenced by packet loss and attenuation.

The detection rate is computed as the ratio of detected attacks to the total number of attacks. The results are represented in Figure 3.



	Atomic	MAC	Flood	Time	Space	Routing
Detected	40	40	35	32	31	40
False Positives	0	0	0	0	3	2
Undetected	0	0	5	8	6	0

Fig. 3. Detected attacks, undetected attacks and false positives.

As expected, all 40 attacks have been correctly detected by the Atomic and MAC engines, since these engines analyze specific fields in the protocol header, and all packets with an invalid field that reach the detector node are identified as malicious.

The lower detection rates of the Time and the Space Correlation engines (80%) are caused by packet loss and delay, which increase the time interval between two consecutive packets, thus preventing attack detection. The Space correlation engine produced 3 false positives, also caused by packet loss, which leads to desynchronized values on the detector node.

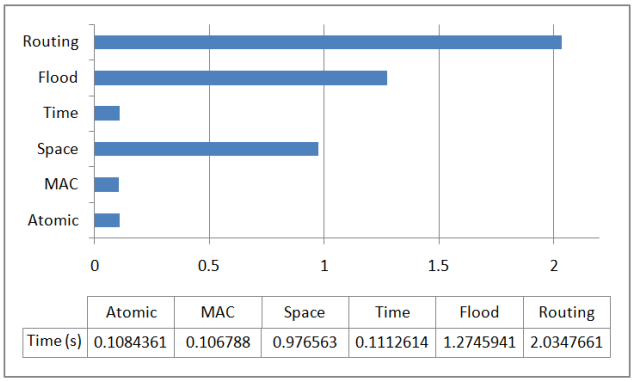
The Flood engine detected 35 attacks out of 40, also due to the loss of attack packets on the path from the attacker to the detector node, which causes lower rate and peak and greater gap values. The Routing engine has detected all attacks, but it has generated 2 false positives because of packet loss.

5.2.2. Detection Time

As mentioned in the previous section, each attack scenario has been run 40 times and considered the detection time. In Figure 4, the average detection time for each attack scenario has been represented.

The lowest detection time has been obtained for the MAC engine, the Atomic engine and the Time Correlation engine, because these engines are able to detect an attack immediately after the malicious packet is received. The Space Correlation engine as a much higher detection time because it has to process several packets (from the nodes in the cluster)

before detecting an attack. The Flood engine has a higher detection time because a number of packets equal to the rate limit must be processed before the signature is triggered. The Routing engine has the highest detection time, because the Blackhole attack is detected after a large number of packets are dropped by the malicious node.



	Atomic	MAC	Space	Time	Flood	Routing
Time (s)	0.1084361	0.106788	0.976563	0.1112614	1.2745941	2.0347661

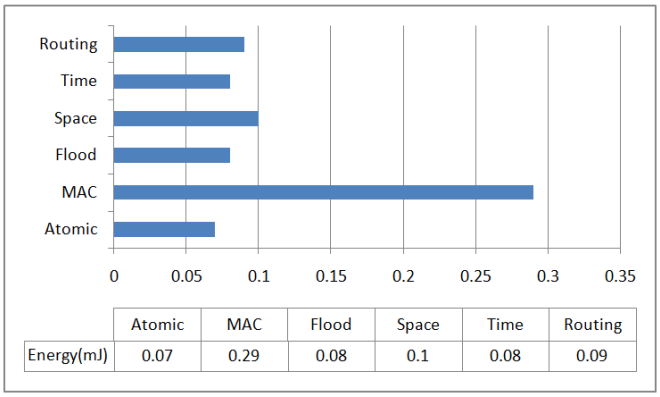
Fig. 4. Detection Time(s).

5.2.3. Energy Consumption

Dedicated tools have been used to evaluate resource consumption on detector nodes. The most important resources in a sensor network are energy and memory.

PowerTOSSIMz has been used to determine the energy consumption for the detector node (Perla et al. 2008). The values are represented in Figure 5.

The most energy efficient engines are the Atomic, Time Correlation, Flood and Routing engines. The most energy consuming engine is the MAC engine because of the HMAC-MD5 algorithm. As a conclusion, TinyAFD is highly energy efficient, even if the most CPU and I/O intensive engines are enabled.



	Atomic	MAC	Flood	Space	Time	Routing
Energy(mJ)	0.07	0.29	0.08	0.1	0.08	0.09

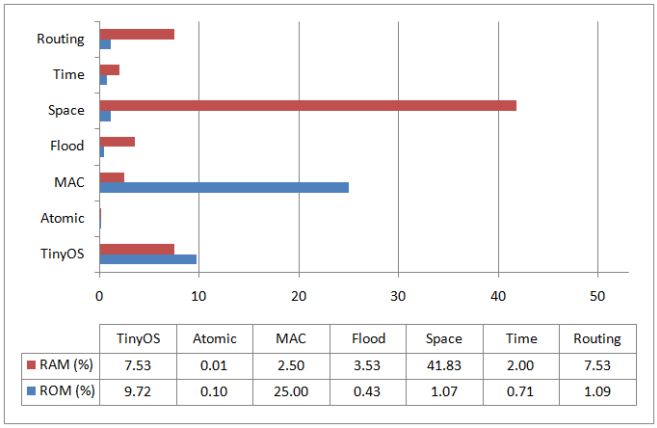
Fig. 5. Energy consumption (mJ).

5.2.4. Memory Usage

A Perl script has been used to determine the RAM and ROM usage by a component from the compiled executable. It is considered the MicaZ platform that has 4 KB of RAM and 128 KB of ROM.

The RAM usage results are presented in Figure 6. The Space Correlation engine occupies the most RAM resources because it stores packets from each conversation that passes through the detector node. The second most RAM consuming is the Routing engine, which stores intercepted packets. The engine that consumes the least energy is the Atomic engine because it does not store any data.

The ROM usage values can also be observed in Figure 6. The most ROM consuming engine is the MAC engine because of the HMAC-MD5 algorithm. After that the Routing and Space Correlation engines occupy the most ROM because of the algorithms they implement. The least ROM consuming engine is the Atomic engine.



	TinyOS	Atomic	MAC	Flood	Space	Time	Routing
RAM (%)	7.53	0.01	2.50	3.53	41.83	2.00	7.53
ROM (%)	9.72	0.10	25.00	0.43	1.07	0.71	1.09

Fig. 6. RAM and ROM usage for TinyAFD engines as % of total MICAz resources.

5.2.5. Impact of TinyAFD

Most critical WSN applications have the requirement of resource-efficiency. (Hill et al. 2000) determined that transmitting one bit consumes as much as executing 1000 instructions. As observed in the previous sections, TinyAFD is very efficient in terms of energy and memory consumption. Resources are consumed when running detection routines, because the framework does not involve communication between neighbor nodes or between detector nodes.

A critical application may also require the minimization of false alerts. The framework produces a small number of false positives. However, as WSNs are usually not accessible from the Internet, there is a small attack rate expected. Therefore, the administrator may invest time in analyzing false alerts by correlating them with the status and position of the nodes. This also can be done automatically, by implementing a mechanism for excluding false alerts.

5.3. Comparative Evaluation

Security solutions similar to TinyAFD are the ones developed by (Silva et al. 2005; Strikos 2007; Stetsko et al. 2010; Hai et al. 2010). The ability to detect specific attacks has been evaluated, as detailed in Table 1.

The solution developed by Silva et al. uses a pre-defined set of rules, which have been also used by Strikos, therefore both solutions are able to detect the same attacks. A replay attack may be detected using a repetition rule and a packet alteration attack using an integrity rule. The Flooding attack may be detected using an interval rule or a repetition rule. Selective Forwarding and Blackhole attacks can be detected using a

retransmission rule. A Delayed Forwarding may be detected using a delay rule. Wormhole and Hello flood attacks may be identified using the radio transmission range rule. Jamming attack is detected using the jamming rule. In most cases the attacker must be in the range of the monitor node, for the attack to be detected.

The solution proposed by (Stetsko et al. 2010) is able to detect attacks through the collaboration of neighbor nodes. It was able to detect Selective Forwarding, Hello Flood and Jamming attacks. As a disadvantage, the communication between the neighbor nodes introduces additional resource consumption and it produces a high number of false positives.

The solution designed by Hai et al. takes the set of rules from da Silva et al. and adapts them for detecting attacks against routing protocols. Packet alteration may be detected using the integrity rule, while the Flood attack using the interval rule. Selective Forwarding, Blackhole and Sinkhole attacks can be detected using the neighbor rule. The Delayed Forwarding may be identified using the delay rule. Wormhole and Hello flood attacks can be detected using radio transmission range rule.

Table 1. Comparative Evaluation of TinyAFD.

Attack	Silva/ Strikos	Stetsko	Hai	TinyAFD
Packet injection (external node)	no	no	no	yes
Packet injection (internal node)	no	no	no	yes
Replay attack	yes	no	no	yes
Packet alteration	yes	no	yes	yes
Sybil attack	no	no	no	yes
Flood attack	yes	no	yes	yes
Selective Forwarding	yes	yes	yes	yes
Delayed Forwarding	yes	no	yes	yes
Blackhole attack	yes	no	yes	yes
Wormhole attack	yes	no	yes	yes
Sinkhole attack	no	no	yes	yes
Hello Flood attack	yes	yes	yes	yes
Jamming	yes	yes	no	no

TinyAFD is able to detect malicious packet injection initiated by external nodes using the MAC engine. Packet injection performed by compromised internal nodes is identified using Time and Space Correlation engines. The MAC engine can also detect packet injection, alteration, replay and Sybil attacks using the appropriate signatures. Flood attacks may be detected by the Flood engine. Selective Forwarding, Blackhole and Delayed Forwarding can be identified by the Routing engine. In addition, with the appropriate signatures, the Routing engine is able to detect Wormhole, Sinkhole and Hello flood attacks.

Silva et al., Strikos and TinyAFD are decentralized and lightweight solutions that do not involve communication between detector nodes, using signatures/rules for detecting attacks. The solution developed by Stetsko et al. and Hai et al. involves communication between the nodes, and therefore it consumes much more energy.

When compared with similar solutions, TinyAFD is *modular*, because the code is organized in engines that use signatures for attack detection. The framework is *configurable* at run-time, as the signatures can be added, fine-tuned, enabled, disabled or removed. The detection mechanism should be adapted to a specific context and threat level, and this is achieved by *fine-tuning* the signatures. TinyAFD is *extensible*, as new engines and signatures can be easily added to the framework.

TinyAFD solution is able to detect a large number of attacks, so it provides *detection generality*, in contrast to most intrusion detection systems, which focus on a small set of specific attacks.

The energy consumption is minimal, as TinyAFD does not involve communication between neighbor nodes. As stated before, executing detection routines on the sensor nodes is much cheaper than sending information to the neighbors.

In conclusion, the proposed framework, TinyAFD, in comparison with other similar solutions, provides the advantages of modularity, configurability, extensibility, detection generality and energy efficiency.

6. CONCLUSIONS

In this paper, a modular, lightweight attack and fault detection framework dedicated to low energy sensor networks, Tiny Attack and Fault Detection framework (TinyAFD) has been proposed.

The framework has a modular design and relies on engines that target specific attacks. Five engines, which are linked in a detection chain: the MAC, Atomic, Flood, Time Correlation, and Space Correlation engines, and a sixth engine that intercepts and analyzes all traffic received and sent by the neighbor nodes: the Routing engine, have been implemented. The solution is extensible, allowing for rapid integration of new specialized engines.

Given that each sensor network has its own security requirements and expected threats, TinyAFD allows administrators to fine-tune the attack definitions. Administrators may reconfigure the system during run-time, using commands to add, modify, and delete signatures from each engine. This feature allows for increased adaptability in situations of restricted access to nodes and limited reprogramming possibilities.

The architecture is a lightweight security solution, optimizing energy efficiency and preventing energy drainage attacks by allowing customization of alert messages. The framework is able to detect a wide range of attacks against sensor networks.

TinyAFD has been implemented in TinyOS, an operating system for sensor networks, and tested with TOSSIM in multiple attack scenarios. The framework has been able to successfully detect external and internal false data injection, packet altering, Flooding, Sybil, Blackhole and Selective Forwarding attacks.

The detection rate, false positives and detection time for each engine have been determined. The MAC, Atomic and Routing engines have the highest detection rate, while the Time and Space Correlation engines have the lowest detection rate. The MAC, Atomic, Flood and Time Correlation engines have the least false positives and the Space Correlation and Routing engines have the most false positives. The Atomic, MAC and Time Correlation engines have the lowest detection time, while the Routing engine has the highest detection time.

The energy and memory consumption for each engine in the framework have been evaluated. The Atomic, Flood and Time Correlation and Routing engines consume the least energy and the MAC engine consumes the most energy. The Atomic engine occupies the least RAM and the Space Correlation engine the most RAM. The Atomic engine occupies the least ROM and the MAC engine occupies the most ROM.

TinyAFD has been compared with other similar solutions in regard to detected and mitigated attacks. The framework detects more attacks than other solutions, so it provides detection generality, while having the comparative advantage of being modular, easily extensive and configurable at run-time and energy-efficient.

As a future work, we want to include an automatic mechanism for excluding alerts regarding dead nodes, and nodes that changed their position and association from one detector to another.

Further research is planned to integrate TinyAFD with learning mechanisms that would dynamically decide the parameter values describing normal variability. The solution may be conveniently hybridized by associating it with a lightweight anomaly-based detection framework, thus enabling an adaptable security architecture for low energy sensor networks.

ACKNOWLEDGEMENTS

This work was supported by the research program EXCEL – Excellence in research by postdoctoral programs in priority domains of a knowledge-based society (Excelenta in cercetare prin programe postdoctorale in domenii prioritare ale societatii bazate pe cunoastere), Contract no. POSDRU/89/1.5/S/62557

REFERENCES

Arazi, B. (2009). Message authentication in computationally constrained environments. *IEEE Transactions on Mobile Computing,* 8, 968–974.

Bardac, M., Milescu, G., Deaconescu, R. (2009), Monitoring a BitTorrent tracker for peer-to-peer system analysis, *Intelligent Distributed Computing III*, 203-208

Bojkovic, Z. S., Bakmaz, B. M., Bakmaz, M. R. (2008). Security issues in wireless sensor networks. *International Journal of Communications*, 2, 106-115.

Benenson, Z., Blaß, E.-O., and Freiling, F. C. (2010). Attacker Models for Wireless Sensor Networks. *it - Information Technology*, vol. 52, no. 6, pp. 320-324.

Da Silva, A. P.R., Martins, M.H.T., Rocha, B.P.S., Loureiro, A.A.F., Ruiz, L.B., Wong, H.C. (2005). Decentralized intrusion detection in wireless sensor networks. *Proceedings of the 1st ACM international workshop on Quality of service & security in wireless and mobile networks*, ACM, New York, pp. 16-23.

Gheorghe, L., Rughiniș, R., Țăpuș, N. (2010a). Attack evaluation and mitigation framework. *Proceedings of the Sixth International Conference on Networking and Services* (ICNS ’10), pp. 243-252.

Gheorghe, L., Rughiniș, R., Deaconescu, R., Țăpuș, N. (2010b). Adaptive Trust Management Protocol Based on Fault Detection for Wireless Sensor Networks. *Proceedings of the Second International Conferences on Advanced Service Computing*, pp. 216-221.

Hai, T.H., Huh, E.N., Jo, M. (2010). A lightweight intrusion detection framework for wireless sensor networks. *Wireless Communications and Mobile Computing*, 10, 559-572.

Heinzelman, W.R., Kulik, J., Balakrishnan, H. (1999). Adaptive protocols for information dissemination in wireless sensor networks. *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking* (MobiCom '99), pp. 174-185, 1999.

Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K. (2000). System architecture directions for networked sensors. *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, vol. 35, no.11, pp. 93-104.

Lee, J., Kapitanova, K., Son, S.H., (2010). The price of security in wireless sensor networks. *Computer Networks*, vol. 54, no. 17, pp. 2967–2978.

Levis, P., Lee, N., Welsh, M., Culler, D. (2003). TOSSIM: accurate and scalable simulation of entire TinyOS applications. *Proceedings of The first international conference on Embedded networked sensor systems* (SenSys ’03), ACM Press, New York, pp. 126-137.

Levis, P., Madden, S., Polastre, J., Szewczyk, R., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., Culler, D. (2004). TinyOS: An operating system for sensor networks. *Ambient Intelligence*, Springer, pp. 115-148.

Perla, E., Cathain, A. Ó, Carbajo, R. S., Huggard, M., McGoldrick, C. (2008). PowerTOSSIMz: Realistic Energy Modelling for Wireless Sensor Network Environments. *Proceedings of the 3rd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, pp. 35-42.

Rassam, M.A., Maarof, M.A. and Zainal A. (2012). A Survey of Intrusion Detection Schemes in Wireless Sensor Networks. *American Journal of Applied Sciences* vol. 9 no. 10, pp. 1636–1652.

Rughiniș, R., Gheorghe, L. (2010). Storm control mechanism in wireless sensor networks. *Proceedings of the 9th RoEduNet IEEE International Conference*, IEEE, pp. 430-435.

Scarfone, K., Mell, P. (2007). Guide to intrusion detection and prevention systems (IDPS). *National Institute of*

Standards and Technology NIST Special Publication 800-94, Gaithersburg.

Stetsko, A., Folkman, L., Matyáš, V. (2010). Neighbor-based intrusion detection for wireless sensor networks. *Proceedings of the 6th International Conference on Wireless and Mobile Communications (ICWMC)*, pp. 420-425.

Strikos, A. A. (2007). A full approach for intrusion detection in wireless sensor networks. *School of Information and Communication Technology, KTH*, retrieved on 02.11.2012 from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.102.3855&rep=rep1&type=pdf.

Vasseur, J. (2011). Terminology in low power and lossy networks, retrieved on 02.11.2012 from http://tools.ietf.org/html/draft-ietf-roll-terminology-06 (Work in progress).

Voinescu, A., Tudose, D.S., Tapus, N. (2010). Task Scheduling in Wireless Sensor Networks, Sixth International Conference on Networking and Services INCS 2010, 12-17.

Wang, Y., Attebury, G., Ramamurthy, B. (2006). "A survey of security issues in wireless sensor networks, *IEEE Communications Surveys & Tutorials*, vol. 8, no. 2, pp. 2-23.

Yan, K., Wang, S., Liu, C. (2009). A hybrid intrusion detection system of cluster-based wireless sensor networks. *Proceedings of the International MultiConference of Engineers and Computer Scientists*, pp. 18-20.

Zheng, J., Jamalipour, A. (2009). *Wireless Sensor Networks: A Networking Perspective*, Wiley-IEEE Press, Hoboken, New Jersey.

Zia, T., Zomaya, A. (2006). Security issues in wireless sensor networks. *Proceedings of the International Conference on Systems and Networks Communication* (ICSNC '06), pp. 1-4.