In recent years, we have witnessed the emergence of Shared Sensor Networks (SSNs) as a core component of cyber-physical systems for diverse applications. As Wireless Sensor and Actuator Networks (WSANs) design starts shifting from application-specific platforms to shared system infrastructures, a pressing research challenge is security. In scenarios involving unprotected hostile outdoor areas, SSNs are prone to different types of attacks that can compromise reliability, integrity, and availability of the sensor data traffic and sensor lifetime as well. In this work we propose a Dynamic Security System to be applied in the shared sensor network context. Its basic feature is the nodes neighborhood monitoring and collaboration (through the use of the Byzantine algorithm) to identify an attack and enhance security. The proposed security system is dynamic since it is able to manage the availability, integrity, and confidentiality of multiple applications according to the current execution context. It is also resilient, since it is able to support the continuous network operation even in the presence of malicious or faulty nodes. Its resilience is achieved for the capacity of gathering information from several nodes, thus inferring the countermeasures using context information.

1. Introduction

Recent advances in microelectromechanical systems and wireless communication technologies have enabled the building of low-cost and small-sized sensors nodes, which are capable of sensing, processing, and communicating through wireless links. Wireless Sensor Networks (WSNs) are composed of tens, hundreds, or even thousands of sensor nodes [1, 2]. Nodes in WSNs commonly rely on nonrechargeable batteries as their energy sources, and the replacement of depleted batteries is not always feasible or desirable. Therefore, a major challenge in the design of WSNs is how to save energy in order to extend the network operational lifetime. Wireless Sensor Networks are commonly used to monitor a wide range of environmental variables, such as temperature, humidity, acceleration, and light among others [3]. These networks can thus be regarded as the interface between the physical world and the world of electrical/electronic devices, as computers and domestic appliances [4]. WSNs communicate the information collected by their sensing devices through their wireless links, thus enabling interaction between people or computers and the surrounding environment [5]. The data gathered by the different nodes is transmitted to one or more sink nodes. Sink nodes are devices endowed with high processing power and with no energy constraints. Such nodes act as the entry points for application requests and also as the gathering points of sensor-collected data [1]. The WSN technology can also encompass the use of wireless actuators, originating the concept of Wireless Sensor and Actuator Networks (WSANs) [5]. Besides the inherent sensing capabilities of the WSNs, the additional
feature endowed by actuators nodes makes WSANs able to
be used for controlling their environment by triggering such
nodes. Actuator nodes are able to convert an electrical signal
(a virtual command) into a physical phenomenon (an action)
as sounding alarms, switched on/off electric appliances, or
closing gates.

In the last few years the field of WSANs has experienced
several changes, which have impacted the design and opera-
tion of such networks. One of the most noteworthy signs of
these changes is the emergence of Shared Sensor Networks
(SSNs) [4], which, instead of assuming a traditional approach
of application-specific network design, allow the sensing
infrastructure to be shared among multiple applications,
potentially belonging to different users/owners. The SSN
scenario may encompass multiple networks with different
users/owners resulting in a totally shared infrastructure,
where the resources (nodes in the SSN) are used by multiple
applications, crossing the frontier of the traditional concept
of specific networks/domains. Therefore, SSNs can be seen
as integrated cyber-physical system infrastructures [5], which
can serve a multitude of applications [6]. One of the suitable
contexts to employ SSNs includes the smart spaces [7],
which encompasses smart grids, smart buildings, and smart
cities [6]. The potential advantage of a shared design is the
significant reduction in the costs of deploying the network
by allowing the multiple applications to make use of the
same nodes and networking infrastructure, thus improving
the resources’ utilization. However, the adoption of such
shared design poses new research challenges. In particular,
in this work we are interested in investigating the security
issue [8–10] in such shared networks. There are several
difficulties inherited from WSN for dealing with security: (i)
the limited sensor resources (in terms of processing, memory
and energy); (ii) vulnerabilities associated with wireless
communication and ad hoc organization; and (iii) the fact
that the sensors are deployed in open, unprotected, and
often hostile areas. Besides all those issues inherent of WSN,
security in SSN has additional challenges that must be dealt
with due to the several requirements of multiple applications
running in the nodes. These requirements may be conflicting
or redundant. SSNs can become targets of a large number of
attacks in order to compromise the confidentiality, integrity,
and availability of its data or to decrease the network lifetime
[11].

One way to provide security for SSN and, consequently,
for the applications running on the network is making
use of a Dynamic Security System. A security system is
required to be dynamic since it must be able to manage
the availability, integrity, and confidentiality of multiple
applications according to the current execution context.
Context information is collected by sensors and/or obtained
through the sink node. According to [12], context is any
information about the execution of an application that can
be used to enhance its behavior to operate in a personal,
autonomous, and flexible way. Therefore, a system is context-
aware when it is able to consider environmental information
to provide information or services to users. In this work we
consider context information as (i) security requirements of
the running applications; (ii) intruder indicator and/or attack
in SSN, where such information is provided by an IDS or
other systems capable of detecting an attack on the network;
(iii) the amount of resources available in the sensor nodes,
such as available memory or residual energy.

From the SSNs’ point of view, it is important to note
that depending on the application that is being executed,
the security requirements may differ. For example, military
applications require strict procedures to ensure the protection
of confidential data, while other applications such as fire
detection in forests may not have the same kind of concern.
Therefore, the level of security, which is information that
defines the security controls that should be used to meet
the security requirements, should be modified as the context
information change. This feature calls for a dynamic, adaptive
behavior for the security system.

To tackle the challenges related to security in SSN, safety
levels should be changed dynamically to mitigate vulnera-
bilities and extend the lifetime of the network preventing
successful attacks and saving energy, since security controls
are activated only when required and meeting the different
application requirements. In this sense, each new application
that enters the network requires the security system to adapt
itself to the new requirements of the application and of the
SSN itself.

Another challenge to be faced concerns how to ensure
that the SSN will still work properly when some of the nodes
fail to deliver correct information about the environment. The
property of a system to remain operating even in the presence
of fail in some of its components is known as resilience.
Therefore, ensuring the functioning of the security system
even in presence of failure means to ensure its resilience [13].
In a highly dynamic environment, incorrect information may
come not only from malicious nodes but also from nodes
without updated information about the environment due to
any malfunctions. Several algorithms have been proposed
[11, 13–15] to guarantee resilience in distributed systems.
Among them the Byzantine fault/failure algorithm [11] trusts
in information exchange to build a consensus among several
nodes to make a decision. In a security system for SSN, the
Byzantine algorithm would ensure that nodes would take
the same decision about changing the security levels even
in the presence of malicious nodes or nodes that have made
incorrect decisions.

In this context, the aim of this work is to provide a
dynamic and resilient security system for SSNs, called DYSSS,
in order to ensure the safety of the sensors and of the
information transmitted in the network, while not increasing
the energy consumption of nodes. DYSSS is dynamic and has
a small overhead since it modifies the security levels while
running according to the current context. In this work, we
use a Byzantine fault/failure algorithm [11], which relies on
the cooperation among sensors to ensure that the security
level is changed disregarding the context information sent by
unreliable sensors.

The remainder of this paper is organized as follows:
Section 2 discusses related work, Section 3 presents the
description of DYSSS, performed experiments and analysis of
results are described in Section 4, and conclusions are drawn in
Section 5.
2. Related Work

The SSN approach has gained momentum only recently. In Efstratiou et al. [6] an extension to the traditional concept of WSANs (which aims at supporting a single application and a single user) is proposed where the network infrastructure is decoupled from the application definitions. A framework is created which allows WSAN infrastructures to be shared among multiple applications, which can potentially be owned by different users. By achieving this level of decoupling, WSAN infrastructures can be viewed as an accessible resource, which can be dynamically repurposed and reprogrammed by different authorities, in order to support multiple applications. In Efstratiou et al. [6] the authors work has focused only on the infrastructure definition and the strategy for sharing the nodes resources. Another work that has followed the same direction was SenShare [16], in which the authors presented an approach for building overlay sensor networks that are responsible for not only providing the most suitable nodes to perform an application, but also isolating the network traffic of a given application from the traffic generated by other applications. In [16] the authors describe the supporting mechanisms that are used to maintain the network overlay. The nodes form several clusters and they are normally isolated from each other. For constructing a SSN from these separated clusters as a single connected logic network, virtual links between the clusters need to be established with the help of nodes that are not performing tasks for the target application. Virtual links between clusters are incrementally generated by three consecutive steps, namely, (1) identify the nodes that are on the edges of a connected node cluster, (2) discover optimum paths from the nodes selected in the previous step that connect the local cluster to other clusters, and (3) ensure all the clusters are connected together and can access the network's sink.

In order to ensure the safety when the applications are performed in SSN, several security mechanisms have been recently proposed. These mechanisms aim at preventing attackers from compromising data sent in critical applications, such as smart grids and smart buildings since attacks on these applications could cause damage and endanger citizens.

He et al. [17] propose a secure and distributed data discovery and dissemination protocol called DiDrip (named after two well-known data dissemination protocols: (i) Dip [18] and (ii) Drip [19]). DiDrip allows the network owners to authorize users with different privileges to simultaneously and directly command sensors to disseminate data items (a tuple containing a cryptographic key, a version, and sensed data) in the SSN. Moreover, as demonstrated by their theoretical analysis, DiDrip addresses the authenticity and integrity of the disseminated data in SSNs.

In [20] the authors address the controlled usage of resources as a primary security requirement in case of sensor sharing. A distributed reference monitor is proposed as an enforcement mechanism for sensor sharing. The monitor is policy-driven which enables lightweight runtime control of accesses to resources. The strategy is basically making the monitor choose the tasks that will better fit in a node without depleting its resources. By doing so, the authors hope to avoid attacks that will deplete the nodes, making the network cease communication. The proposal is an extension of the work in [21] and consists of an operational model that inserts controls in the network by instrumentation of local or remote interaction in the resource-rich backend (such as in a testbed), subsequently enforcing control at the nodes by using policy engines. The solution achieves local resource protection and protection of distributed event-based data flow.

Maerine et al. [22] presents SASHA: a proof-of-concept protocol that enables lightweight and secure deployment of multiple applications on heterogeneously owned sensor nodes. The protocol requires an Application Owner (AO) to request permission from all the Platform Owners (POs) on whose infrastructure he/she wishes to install an application. Each PO validates this request and, if approved, grants a token which the AO can use to deploy his/her application. This token also contains parameters to limit the amount and frequency a component can send and the CPU time it can use.

In [23] the authors present the SecLooCI WSN middleware (a continuation of the work begun in SASHA), which enables secure multiparty interactions on top of resource constrained sensor nodes. In this work five key data flows that need additional support from the node middleware to ensure safe and secure multiplexer node interactions are identified: (1) network initialisation, (2) application deployment, (3) application management, (4) application service usage, and (5) application communication. Based on the flows, the authors propose new components to deal with each of these challenges.

In [24], the authors propose a dynamic instruction detection system for SSNs called DISSN. DISSN gathers information about the environment (packet loss, interference, and system lifetime) as indications of attacks in the SSN. According to the information gathered, DISSN will choose a countermeasure, based on the QoS requirements of an application. As an example, a high rate of packet loss could be an indication of Denial of Service and a countermeasure to change the node state to idle.

Although several efforts have been made in order to ensure safety of SSN security, none of them but DISSN [24] are resilient to faulty nodes. Although, DISSN is not context-aware, being dynamic only in the sense that new applications may arrive during its operation. DYSSS is context-aware and adapts itself according to the network conditions.

3. DYSSS: A Dynamic and Context-Aware Security Service for Shared Sensor Networks

The aim of this work is to propose a resilient Dynamic Security System for SSNs, called DYSSS, capable to provide a protected environment for SSN applications, without wasting resources inappropriately, especially energy. For achieving its goal, DYSSS dynamically adjusts the SSN security controls in order to satisfy the requirements of its applications, at runtime, according to the changes in the execution context. By context, we mean requirements that each application has regarding cryptography, loss tolerance (how tolerant an
application is to packet loss), and delay tolerance, as well as information about the network such as delay and RSSI (received signal strength indicator). Also we use the residual energy of the node.

3.1. Logical Architecture. Figure 1 presents DYSSS architecture, which encompasses the following components: Application Manager, Context Manager, Dynamic Security Manager, Security Rules Manager, Resilience Manager, Adjustment Manager, and Network Manager. In addition, this subsystem comprises two databases: Context Information, Application, and Security Rules.

The Application Manager is responsible for managing the arrival of applications in the SSN, which means to insert in the Application database the application IDS, the tasks that are to be executed, and security requirements. The Context Manager component is responsible for updating the Context Information database and notifying the Dynamic Security Manager whenever the context changes. For this, the Context Manager gathers both internal context information (such as the residual energy from the sensor node) and external information (i.e., from an external network, such as new settings given by theSink node). The Dynamic Security Manager component is responsible for conducting the action performed by DYSSS to satisfy the security requirements of the applications according to the current context. The Security Rules Manager component is responsible for processing the security rules in order to set the sensor with the suitable security level for the current context. It is also responsible for evaluating if there is redundancy in requirements of multiple applications. Whenever two applications share common requirements that trigger a common rule present in the Security Rules database, this rule is evaluated only once. The Resilience Manager is the component responsible for the resilience of DYSSS. For this, it uses the Byzantine fault algorithm in order to ensure consensus when a decision to modify the security level is taken. The Adjustment Manager is the component that sends the command for activating and deactivating the security controls in order to change the security level. The Network Manager component is responsible for exchanging messages among all sensors running DYSSS. This component is used specifically for transmitting messages used to perform the Byzantine consensus, such as described in [9].

As previously mentioned, DYSSS has three databases: Context Information, Application, and Security Rules. The Context Information database stores the most recent information about the context. This base is organized in the following fields: (i) the security requirement of the application (represented by an 8-bit integer), where, for this work, we considered packet loss, delay, RSSI, and cryptography algorithm; (ii) a value that identifies the existence of intrusion or attacks; (iii) the residual battery power of sensors. The Security Rules database stores the rules that define how the security level must be changed. These rules are arranged in the following format: IF <condition> THEN <security level>, allowing evaluation of the current context under some predetermined threshold before taking a decision. For instance, IF PACKETLOSS <= 0.7 AND residual battery power > 90% THEN change the security level to 3. Each level uses different cryptography algorithms, verification rate of the security system, and number of nodes participating in the Byzantine consensus, with security levels 0 being without any security procedure and 5 the most restrict. The Application database contains the application identifier (16-bit integer), types of sensors required (sensing interfaces, 8-bit integers), sensing rates and the frequency (in milliseconds) with which the decision process of each application is triggered, and the identifiers for the tasks (one 16-bit integer) that the framework can perform. These tasks are methods implemented in the framework and they are divided into 3 groups: reading, decision, and sending messages. The reading tasks represent the collection of data from the monitored environment; the decision task involves decisions taken for the applications (the results of applying the rules stored at Security Rules database); sending messages that represent the exchange of messages between the nodes in the SSN. The security requirements can be seen as a tuple considering {delay in percentage, packet loss in percentage and residual battery power).
3.2. DYSSS Operations. As soon as the application deployment is finished, DYSSS starts its operation. The Context Manager periodically checks the information from the sensor (internal context) and the information collected by the sensors (external information) and updates the context information. Whenever any change occurs in the context, the Context Manager updates the Context Information database and notifies the Dynamic Security Manager that the context has changed in order to activate the other operations of DYSSS. The Context Manager gives an order to the Application Manager for it to search in the Applications database for the information regarding the applications in the networks. If a new application arises, the Application Manager will store the new application parameter in the application database. Then, Dynamic Security Manager sends the new context information for the Security Rules Manager to determine what actions should be taken. Upon receiving the new context information, Security Rule Manager processes it together with information from Security Rules database in order to decide on the appropriate security level for a given application. If the new recommended security level is different from the current security level, the Dynamic Security Manager activates Resilience Manager to validate the decision and change the security controls. After receiving a decision to change the security level, the Resilience Manager requests the Network Manager to send a message to the other sensors. Then Resilience Manager waits for a certain number of messages to check whether there is consensus to change the security level. In order to find consensus we use a Byzantine algorithm [14]. This Byzantine algorithm seeks to find a consensus among the decisions taken by different nodes. This algorithm works only when the number of nodes “N” is at least 3\(M + 1\) where “M” is the number of nodes that send a different decision. Each node exchanges messages with the other DYSSS nodes in the network. One node sends a decision to \(N − 1\) other nodes. Then, each receiver node communicates the received value to the other \(N−2\) nodes, and so on. When all nodes have sent their decisions, each node takes the majority (half of the number of DYSSS nodes \(+1\)) of the decisions received from the other DYSSS nodes and uses that as the final decision for that changing security level.

If there is consensus among the nodes, the Resilience Manager triggers the Adjustment Manager in order to make necessary modifications in the security controls. By modifications we mean to change to a new security level so as to meet the current execution context and ongoing application requirements.

3.3. Case Study. In a given SSN two applications coexist, namely, a fire detection application and a HVAC (Heating, Ventilation, and Air Conditioning) application. Those applications have different security levels, the fire application having higher security requirements than the HVAC application. In order to ensure the well-functioning of these two applications, DYSSS was installed in some nodes of the network. Two attacks were considered: (i) Sink Hole Attacks [11] and (ii) Sinkhole attacks [25]. Jamming attacks can be characterized by the increasing number of packets in the network and node’s energy depletion according to [25]. Sinkhole attacks (black hole and selective forwarding) can be characterized by degrading the RSSI and increasing packets loss according to [11].

If compromised nodes start an attack (either jamming or a Sinkhole attack), the nodes with DYSSS will update the Context Information Database with the most recent information about the environment (in terms of packet loss, delay, RSSI, and residual energy). This database stores the last information about the environment, the residual energy of the node, and the security requirements of the applications. An example of security requirements stored in this database is for HVAC application 50% of packet loss, 10% of RSSI, 50 ms of delay, and no cryptography while for a fire detection application it is 5% of packet loss, 80% of RSSI, 25 ms of delay, and AES cryptography. If any event in the environment has triggered a change in the context, the Context Manager sends the new context information (e.g., 15% of packet loss, 85% of RSSI, and 78% of residual energy) to the Dynamic Security Manager. Such component, in its turn, will query the Security Rules Manager if there is some rule adequate to this new context information in the Security Rules database (e.g., there can be a rule in this database such as IF packet loss > 10% then security level = 1). If there is a change in the security level, this decision is exchanged among the other nodes using DYSSS using a Byzantine Algorithm through the Resilience Manager (it sends a message containing the node ID, the applications ID, and the new security level). If after the Byzantine consensus the decision is to change the level, the Adjustment Manager changes the security controls. As an example, if the security level is 1 then start to use an acknowledgement protocol.

4. Experiments and Performance Evaluation

In this section, we present the experiments performed to evaluate DYSSS. In Section 4.1, we present the details of the implemented prototype. In Section 4.2, we describe the scenarios and metrics used in the experiments. Section 4.3 presents DYSSS calibration. Sections 4.4 and 4.5 present experiments about DYSSS overhead and efficacy, respectively. In Section 4.6 a comparison with another existing approach is described. Finally in Section 4.7, a comparison between results achieved with the implementation of DYSSS in real nodes and its operation in simulated nodes is performed.

4.1. Prototype. In all experiments performed with DYSSS, the SSN is composed of Sun Small Programmable Object Technology (SUN Spot) sensors. The SUN Spot platform features an ARM 32-bit processor operating at a frequency of 400 MHz; 1 MB of RAM; 4 or 8 MB of Flash memory; and Chipcon 2420 radio for communication in adhering to the IEEE 802.15.4 protocol featuring 11 channels of 2.4 GHz [26]. DYSSS has been implemented using the programming language Java 2 Micro Edition (J2ME) with SUN Spot development kit. The simulation scenarios were performed using Solarium simulator [26], which contains a SPOT emulator that is useful for testing SPOT software and/or to create scenarios with a large number of nodes whenever the real hardware is not available. In the performed experiments, we
created a mix of real nodes with emulated SPOTS by using Solarium so that we can assess the system scalability.

The SUN Spot development kit provides several software components, including those that implement the stack of network communication protocols. DYSSS was implemented by defining new components capable of receiving context information and making the decision that changes the security controls in order to adjust (increase/decrease) the required security level. In order to simplify our prototype, we have inserted all possible applications at deployment time in the applications database. Sensors running DYSSS followed the most demanding application data rate in the network, since the security system had to respond in all situations (meaning, attend requirements of all running applications). Fire detection applications are high demanding applications in terms of data rate: according to [24] a message must be sent every 10 ms. In order to simplify and to prove that DYSSS could work under demanding requirements, we have used 10 ms as DYSSS data rate.

4.2. Experiment Environment and Metrics. We have performed 4 sets of experiments with the following goals: (i) to evaluate the overhead of DYSSS in terms of memory consumption and system lifetime, (ii) to evaluate the efficacy of DYSSS in detecting possible attacks to the SSN, (iii) to compare DYSSS with other security system presented in literature, in this case Conti et al. [15], and (iv) to compare DYSSS with other security system presented in literature, in this case Conti et al. [15], and (iv) to compare results achieved with real and simulated nodes in order to prove the practical feasibility of DYSSS. The first three experiments used simulated nodes and the last one used both real and simulated nodes.

In our simulations, we adopted a flat network topology with static nodes installed in the environment. We varied the number of nodes in which DYSSS was deployed from 20% to 100% of nodes (Section 4.3). The number of nodes in the SSN varied from 50 to 500 nodes (50, 100, 250, and 500), in order to evaluate DYSSS scalability, having been arranged in the same rows equidistant from each other varying from 100 m × 100 m square space (100000 m² - 50 nodes) to 317 m × 317 m (100000 m² - 500 nodes). The area size varied in order to maintain node density. During the simulation, compromised nodes were configured to act as if it was compromised at different periods in order to provide inaccurate context information, meaning that they would send incorrect security decisions over attacks. All positions of the nodes were static. All nodes that contain DYSSS were previously chosen. All compromised nodes were previously chosen.

The experiments lasted for 3600 seconds (1 hour). Finally, we consider the confidence interval of 95% with satisfactorily strict limits after repeating each simulation 30 times. We have varied the number of applications in the network from 2 to 10 applications (2, 4, 6, 8, and 10 applications) in the SSN in all experiments but in those described in Sections 4.5 and 4.7. These applications (meaning sensing types and rates) as well as their security levels were generated through a random generation procedure, further explained in [27]. It is discussed in the literature that randomization may not always represent real applications; however, the diversity they provide is sufficient for this group of experiments as explained in [27].

DYSSS overhead was measured using system lifetime as metric regarding energy consumption. It was evaluated using SUN SPOT API. We have also evaluated memory consumption (in Kbytes), also through SUN SPOT API.

DYSSS efficacy relates to its capability of correctly choosing the security level. In order to measure efficacy, we have used True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). FP indicates DYSSS deciding to change security level when it should not and FN DYSSS staying in a security level when in fact it should change. TP indicates that DYSSS changed to a correct security level and TN that DYSSS correctly stayed and the same security level. The value obtained by the sum TP + TN was denominated precision.

4.3. DYSSS Calibration. The purpose of this experiment is to define the percentage of nodes in the SSN that should include DYSSS. We have made this experiment with 500 nodes and 10 applications in the SSN. We varied the number of nodes with DYSSS from 100 to 500 (100, 200, 300, 400, and 500). As we can see in Table 1, the precision of DYSSS remains the same after 20% of the nodes containing DYSSS. In this sense, it is better to maintain DYSSS in 20% of the nodes to reduce energy consumption. Even though the nodes have a small vision of the whole network, the fact that they exchange messages with one another makes them able to maintain the same precision value even with only 20% of nodes.

4.4. Overhead. Regarding memory consumption, DYSSS components and databases consumed 26 Kbytes in size, consuming additional 50 Kbytes during its execution. Regarding the system lifetime, using DYSSS, we varied the number of nodes from 50 to 500 and the number of applications from 2 to 10 apps. The results are shown in Table 2. Results shown in Table 2 indicate that, with the increment in the number of applications running in the SSN, the system lifetime dropped as expected. More importantly, as expected, the relationship

| Number of nodes | System lifetime (days) |
|-----------------|------------------------|
| 50              | 28 128 114 108 98      |
| 100             | 51 122 108 102 94      |
| 250             | 12 116 107 99 92       |
| 500             | 9  108 103 93 89       |

Table 1: DYSSS calibration.

| Percentage of nodes containing DYSSS (%) | 20% | 40% | 60% | 80% | 100% |
|-----------------------------------------|-----|-----|-----|-----|------|
| TP                                      | 28  | 26  | 22  | 31  | 29   |
| TN                                      | 51  | 54  | 58  | 48  | 50   |
| FP                                      | 12  | 10  | 12  | 8   | 11   |
| FN                                      | 9   | 10  | 8   | 13  | 10   |
between the number of applications and the system lifetime is not linear. This behavior is due to the fact that as more applications are used this naturally increases the possibility of finding common rules between them. Therefore, DYSSS is able to fully utilize the common rules in order to reduce the system energy consumption by executing those rules only once.

4.5. Efficacy. Two simulation experiments were conducted to evaluate if DYSSS is able to properly ensure the safety of the network even in the presence of an attack or some malfunction of a node. In the first experiment, the resilience manager component is disabled, while in the second experiment this component is enabled. In these experiments, we have used 500 nodes and 10 applications. In both experiments, 20% of the sensor nodes were configured as compromised nodes. Also, in both experiments, during part of the nodes lifetime, a compromised node was sending incorrect decisions about the security level that the SSN must have to satisfy the security requirements of the running applications. The time period the compromised node was sending incorrect decisions varied from 0% to 100% of the total time period of the experiment. Each compromised node would send at first the incorrect messages then it would behave normally.

Table 3 shows how DYSSS capacity of detecting anomalies on the network is affected by the percentage of false information sent by the compromised nodes. We observe that, as expected, as the percentage of inaccurate decisions increases, the system becomes less accurate in taking security decisions (precision will eventually decrease to 0%).

With the activation of the Resilience Manager component, even in the worst case (the compromised node sending incorrect information during 100% of the time), DYSSS was able to ensure a secure system operation, due to the Byzantine failure algorithm [28] as seen from the results in Figure 2.

4.6. Comparison. In order to verify the efficacy of the proposed security system, we compared DYSSS with the work presented by Conti et al. [15].

All conducted experiments described in this section adopted the same scenario in terms of topology, in terms of number of nodes present in the SSN, and in terms of the functioning of the compromised node. The nodes were kept static throughout the simulation. The compromised node operated normally until the attack starts. The scenario consisted of 50 sensor nodes randomly distributed over an area of 100 × 100 square meters. Each of these sensors had a sensing range of 50 meters radius. We have reimplemented the work of Conti et al. [15] using Sun SPOT/Solarium. In this set of experiments, we have varied the number of applications in the network from 2 to 10 varying by 2 in each experiment and compared DYSSS and Conti et al. [15] in terms of System Lifetime and Efficacy.

Regarding system lifetime, the main goal of this experiment is to assess how long the SSN would last under the given number of applications. Results shown in Table 4 indicate that, with the increment in the number of applications running in the SSN, the system lifetime is dropped. More importantly, as expected, the relationship between the number of applications and the system lifetime is not linear. As more applications are used as input this naturally increases the possibility of finding common rules between them; thus the proposed algorithm is able to fully utilize the rules to reduce the system energy consumption by executing those rules only once.

Regarding efficacy, as previously mentioned, this metric refers to the percentage of attacks that are correctly detected and then triggering a successfully performed countermeasure. If an attack is not detected, it might compromise the entire SSN operation. We verified the efficacy by varying the number of applications in the system, as shown in Table 5. We

Table 3: DYSSS efficacy in terms of correct decisions.

| % of correct decisions sent by the compromised sensor | 100% | 80% | 60% | 40% | 20% | 0% |
|---|---|---|---|---|---|---|
| VP | 47 | 38 | 34 | 33 | 32 | 30 |
| VN | 53 | 51 | 38 | 34 | 16 | 10 |
| FP | 0 | 7 | 9 | 14 | 29 | 19 |
| FN | 0 | 4 | 19 | 19 | 23 | 41 |

Figure 2: Efficacy after consensus.

Table 4: System lifetime comparison.

| System lifetime (days) | 2 apps | 4 apps | 6 apps | 8 apps | 10 apps |
|---|---|---|---|---|---|
| DYSSS | 137 | 128 | 114 | 108 | 98 |
| Conti et al. [15] | 102 | 84 | 68 | 53 | 37 |

Table 5: Efficacy comparison.

| Efficacy (%) (precision) | 2 apps | 4 apps | 6 apps | 8 apps | 10 apps |
|---|---|---|---|---|---|
| DYSSS | 80% | 76% | 70% | 69% | 70% |
| Conti et al. [15] | 70% | 56% | 44% | 43% | 34% |
can observe that when the number of applications increases, the efficacy for both algorithms decreases. DYSSS can guarantee a precision of more than 70% under all experiments conditions, but Conti et al. failed to deliver such correct decisions rate since it is not resilient. DYSSS is resilient due to the consensus among its nodes for identifying incorrect decisions coming from malicious/malfunctioning nodes.

4.7. Experiments with Real Nodes. In this section, the same scenario simulated using Solarium was implemented on a real sensor node platform. We aimed to validate the results obtained from simulations by comparing them with the results obtained from a real SSN setting. This real experiment was performed in a controlled environment (our research laboratory in UFRJ). In this case the nodes were kept stationary and disposed on the floor of the lab. We have used the same scenario used in Section 4.4.

The obtained average results in the experiments with a real sensor node for System lifetime (134 days) can be considered similar to those in the simulated experiment (137 days). Considering the confidence interval of 95%, the observed differences between the obtained results in the real and simulated experiments were due to imprecisions in Solarium and in real experiments. Therefore, these results validate the experiments performed in the simulated environment. In terms of efficacy, the simulated experiment achieved slightly better results than the real experiment, as it can be seen on Table 6. Such difference can be explained by the occurrence of interferences on the real experiment that induced some packet loss that could not be matched by the noise simulation component of Solarium.

5. Conclusions

This paper presented a Dynamic Security System for SSN called DYSSS, whose main objective was to dynamically meet the security requirements of applications according to the execution context. With DYSSS it is possible to ensure the security requirements in a dynamic and heterogeneous environment, where security controls are different depending on the current context. The use of the proposed DYSSS brings the following benefits: (i) allowing the security to be adjusted according to the current context where applications are executed, thereby allowing the limited resources of sensor nodes to be used efficiently, and (ii) ensuring, through the consensus among the nodes in the SSN, that even in the occurrence of compromised nodes the decision of security adjustments is carried out correctly.

In future work, we intend to investigate other security system options, in order to analyze the relationship between energy expenditure and efficiency of the security systems resilience. We also intend to study techniques for solving conflicts among rules and, finally, to develop a technique to dynamically learn about new attacks in the network and create rules for them based on the experience of previous attacks.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is partly supported by the Brazilian Funding Agencies CNPq and Inmetro. Luci Pirmez, Flávia Coimbra Delicato, and Luiz Fernando Rust da Costa Carmo are CNPQ fellows.

References

[1] Delicato, F. Protti, L. Pirmez, and J. F. de Rezende, ”An efficient heuristic for selecting active nodes in wireless sensor networks,” Computer Networks, vol. 50, no. 18, pp. 3701–3720, 2006.
[2] I. Dietrich and F. Dressler, ”On the lifetime of wireless sensor networks,” ACM Transactions on Sensor Networks, vol. 5, no. 1, article 52, 2009.
[3] K.-C. Tsai, J.-T. Sung, and M.-H. Jin, ”An environment sensor fusion application on smart building skins,” in Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC ’08), pp. 291–295, Taichung, Taiwan, June 2008.
[4] I. Leontiadis, C. Efstratiou, C. Mascolo, and J. Crowcroft, ”Sen-Share: transforming sensor networks into multi-application sensing infrastructures,” in Wireless Sensor Networks: 9th European Conference, EWSN 2012, Trento, Italy, February 15–17, 2012. Proceedings, G. P. Picco and W. Heinzelman, Eds., vol. 7158 of Lecture Notes in Computer Science, pp. 65–81, Springer, Berlin, Germany, 2012.
[5] F.-J. Wu, Y.-F. Kao, and Y.-C. Tseng, ”From wireless sensor networks towards cyber physical systems,” Pervasive and Mobile Computing, vol. 7, no. 4, pp. 397–413, 2011.
[6] C. Efstratiou, I. Leontiadis, C. Mascolo, and J. Crowcroft, ”A shared sensor network infrastructure,” in Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys ’10), pp. 367–368, ACM, Zürich, Switzerland, November 2010.

| % of correct messages sent by a compromised sensor, simulated/real |
|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
| %  | 100% | 80% | 60% | 40% | 20% | 0% |
| TP  | 48/46 | 30/29 | 37/36 | 13/13 | 3/3 | 0/0 |
| TN  | 52/51 | 50/48 | 23/23 | 27/25 | 7/6 | 0/0 |
| FP  | 0/2 | 13/14 | 12/14 | 28/28 | 51/51 | 32/32 |
| FN  | 0/1 | 7/9 | 28/28 | 32/34 | 39/40 | 68/68 |
[7] C. de Farias, H. Soares, L. Pirmez et al., “A control and decision system for smart buildings using wireless sensor and actuator networks,” European Transactions on Telecommunications, vol. 25, no. 1, pp. 120–135, 2014.

[8] J. Laprie, “From dependability to resilience,” in Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN’08), Anchorage, Alaska, USA, June 2008.

[9] X. Jiang, W. Hu, S. Zhu, and G. Cao, “Compromise-resilient anti-jamming communication in wireless sensor networks,” Wireless Networks, vol. 17, no. 6, pp. 1513–1527, 2011.

[10] C. Karlof and D. Wagner, “Secure routing in wireless sensor networks: attacks and countermeasures,” Ad Hoc Networks, vol. 1, no. 2, pp. 293–315, 2003.

[11] H. M. Salmon, C. M. De Farias, P. Loureiro et al., “Intrusion detection system for wireless sensor networks using danger theory immune-inspired techniques,” International Journal of Wireless Information Networks, vol. 20, no. 1, pp. 39–66, 2013.

[12] S. Hoh, J. S. Tan, and M. Hartley, “Context-aware systems—a primer for user-centred services,” BT Technology Journal, vol. 24, no. 2, pp. 186–194, 2006.

[13] S. Sundaram, S. Revzen, and G. Pappas, “A control-theoretic approach to disseminating values and overcoming malicious links in wireless networks,” Automatica, vol. 48, no. 11, pp. 2894–2901, 2012.

[14] M. de Lima and F. G. P. Greve, “Detectando Falhas Bizantinas em Sistemas Distribuídos Dinâmicos,” Revista Brasileira de Redes de Computadores e Sistemas Distribuídos, vol. 2, no. 1, pp. 9–21, 2009.

[15] M. Conti, R. Di Pietro, and A. Spognardi, “Wireless sensor replica detection in mobile environments,” in Proceedings of the 13th International Conference on Distributed Computing and Networking (ICDCN’12), Hong Kong, January 2012.

[16] I. Leontiadis, C. Efstratiou, C. Mascolo, and J. Crowcroft, “SenShare: transforming sensor networks into multi-application sensing infrastructures,” in Wireless Sensor Networks: 9th European Conference, EWSN 2012, Trento, Italy, February 15–17, 2012. Proceedings, vol. 7158 of Lecture Notes in Computer Science, pp. 65–81, Springer, Berlin, Germany, 2012.

[17] D. He, S. Chan, M. Guizani, H. Yang, and B. Zhou, “Secure and distributed data discovery and dissemination in wireless sensor networks,” IEEE Transactions on Parallel and Distributed Systems, vol. 26, no. 4, pp. 1129–1139, 2015.

[18] K. Lin and P. Levis, “Data discovery and dissemination with DIP,” in Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN ’08), pp. 433–444, IEEE, St. Louis, Mo, USA, April 2008.

[19] G. Tolle and D. Culler, “Design of an application-cooperative management system for wireless sensor networks,” in Proceedings of the 2nd European Workshop on Wireless Sensor Networks (EWSN’05), pp. 121–132, IEEE, Istanbul, Turkey, February 2005.

[20] C. Huygens, N. Matthys, and W. Joosens, “Optimized resource access control in shared sensor networks,” in Security and Privacy in Mobile Information and Communication Systems, vol. 47 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pp. 105–116, Springer, Berlin, Germany, 2010.

[21] C. Huygens, N. Matthys, and W. Joosens, Optimizing Resource and Data Security in Shared Sensor Networks, Security and Communication Networks, John Wiley & Sons, Heidelberg, Germany, 2011.
