Distributed Intrusion Detection Systems for enhancing Security in Mobile Wireless Sensor Networks*

Leonardo Mostarda
Dipartimento di Informatica
Universitá degli studi dell'Aquila, Italy.
mostarda@di.univaq.it

Alfredo Navarra
Dipartimento di Matematica e Informatica
Universitá degli Studi di Perugia, Italy.
navarra@dipmat.unipg.it

Abstract

We present an approach to provide Intrusion Detection Systems (IDS) facilities into Wireless Sensors Networks (WSN). WSNs are usually composed of a large number of low power sensors. They require a careful consumption of the available energy in order to prolong the lifetime of the network. From the security point of view, the overhead added to standard protocols must be as light as possible according to the required security level. Starting from the DESERT tool [14, 16, 25] which has been proposed for component-based software architectures, we derive a new framework that permits to dynamically enforce a set of properties of the sensors behavior. This is accomplished by an IDS specification that is automatically translated into few lines of code installed in the sensors. This realizes a distributed system that locally detects violation of the sensors interactions policies and is able to minimize the information sent among sensors in order to discover attacks across the network.

1 Introduction

A Wireless Sensor Network (WSN) usually consists of a number of different modalities sensors that, when combined with a microprocessor and a low-power radio transceiver, form smart network-enabled nodes. The sensed data can be related to different applications but in terms of capabilities all the nodes cooperating in the WSN can be assumed as homogeneous. A first level of cooperation may be viewed just to route collected data outside the sensed area to a fixed infrastructure (the sink nodes) where it is processed. Concerning the applications, medical services, battlefield operations, crisis response, disaster relief, environmental monitoring, premises surveillance, robotics are included. Due to the critical environment where such kind of networks may be used, as a second level of cooperation we are interested in investigating security issues. Such kind of networks are in fact frequently subject to attacks by malicious intruders. Intrusions take place through a sequence of actions that aim to subvert the network system. There are many vulnerabilities and threats to a mobile WSN. They include outages due to equipment breakdown and power failures, non-deliberate damage from environmental factors, physical tampering, and information gathering. In the following we summarize some of them (see [27] for an extended survey).

1. *Passive Information Gathering:* When communications are in clear or an intruder has got in some way the key for decryptions, it can easily pick off the data stream.

2. *Traffic Analysis:* Although communications might be encrypted, an analysis of cause and effect, communications patterns and sensor activity might reveal enough information to enable an adversary to defeat or subvert the mission of a WSN.

3. *Compromised Node:* Due to an external intervention, a sensor may be compromised and can be used to subvert the correct WSN behavior.

4. *False Node:* Additional fake nodes could be thrown in the sensed area sending false data or blocking the passage of true data.

5. *Node Malfunction or Outage:* A node in a WSN may malfunction and generate inaccurate or false data or it could just stop functioning hence compromising used paths.

6. *Message Corruption:* Attacks against the integrity of a message occur when an intruder inserts itself between the source and the destination and modifies the contents of a message.

7. *Denial of Service:* A denial of service attack may take several forms. It may consist in jamming the radio link or it could exhaust resources or misroute data (see [17] for a more detailed discussion).

The category of attacks outlined in 1. and 2. are *passive*

*The research was partially funded by the European project COST Action 293, "Graphs and Algorithms in Communication Networks" (GRAAL). Preliminary results contained in this paper appeared in the [15].

attacks, while the remaining ones are called *active attacks*. This is due to the fact that the former ones do not produce a modification of the traffic network. They are usually addressed by encryption mechanisms that obscure the traffic to unauthorized sensors. A lot of work has been done in this direction in order to reduce the complexity of the applied algorithms and hence the consumed energy (see for instance [5, 6]). In this paper we are interested in the second kind of attacks where the detection of malicious behaviors can be locally detected by observing the traffic over the network. We focalize our attention on Intrusion Detection Systems (IDSs) that analyze the observable behaviors of the systems (see for instance [28, 10, 26, 14, 19]) in order to detect attacks.

Intrusion detection techniques are successfully applied to wired networks, however most of them are not suitable for the wireless context. The inadequacy of standard IDS in WSNs is a consequence of the open medium access, of the dynamic topology, of the cooperation needed among sensors (collaborative algorithms) and the lack of a fixed topology where all information flows. The detection of an attack may not be possible only considering the traffic locally sent/received by a sensor. Therefore, there must be some form of correlation among the data received by the sensors. This will permit to discover attacks that are scattered over several sensors.

Besides the problems mentioned above, in mobile WSNs the detection has to address the problems imposed by the limited battery, restricted computational power of the sensors, low bandwidth and slower links. Moreover, sensors mobility can complicate the detection since the correct behavior of sensors is location dependent (i.e., the sensors correct behavior can change over time). In summary, an IDS for a mobile WSN should: (i) be decentralized; (ii) minimize the traffic overhead; (iii) address the mobility problem.

We use a specification-based approach to provide intrusion facilities into mobile wireless sensor networks. *Specification-based IDSs* [14, 19, 26] use some kind of formal specification to describe the correct behavior of the system. The detection of violations involves monitoring deviations from the formal specification, rather than matching specific well-known attacks. The advantage of this approach is the ability to detect previously unknown attacks at the expense of providing a formal specification of correct information flows.

In this work we automatically generate distributed-specification-based IDSs for mobile WSNs. In our system model each sensor s plays a role r that defines the possible messages sent and/or received by s. We use an automaton (in the following referred to as *Global Automaton*) as the basis in building a specification-based IDS. The Global Automaton defines an ordering among the roles messages and

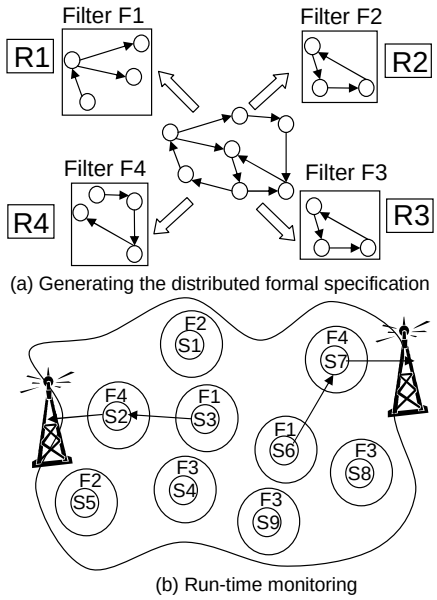

Figure 1. Specification and monitoring

their correct format. We have developed an algorithm that given a Global Automaton produces a set of local automata each of them assigned to a role. For instance, in Figure 1.(a) we draw a Global Automaton decomposed into four local automata each of them assigned to a role. A local automaton (related to a role r) constitutes the basis in building a filter that can be used to monitor each sensor playing the role r. For example, in Figure 1.(b) we show nine sensors $\{S1, \ldots, S9\}$ each of them playing a different role among $\{1, \ldots, 4\}$ and monitored by the related filter $\{F1, \ldots, F4\}$. Each filter locally monitors the messages sent/received by its sensor and locally detects violation of the policy expressed by the Global Automaton.[1]

In order to validate and to explain in more detail this approach we apply it to a recent proposed routing protocol, CoP [23], for mobile WSNs. Note that, while the formal specification directly derives from the considered protocol, the method is completely independent of it. In fact, it can be used on any kind of protocol for mobile WSNs in order to add security aspects.

1.1 Related Work

In the field of intrusion detection systems on mobile WSNs few works appeared in the literature. The mobility issue is typically not addressed and moreover specific kind of attacks are considered separately. In [8], an anomaly approach based on self-organized criticality (SOC) and Hidden Markov models to detect data inconsistencies is pro-

[1] Indeed we observe the interactions with its transmitting/receiving system, enabling it according to the specified polices.

posed. This approach is developed based on the structure of naturally occurring events. With the acquired knowledge distilled from the self-organized criticality aspect of the deployment region, it applies a hidden Markov model. It lets sensor networks adapt to the dynamics norm in its natural surrounding so that any unusual activities can be singled out. Another paper formulates the attack-defense problem by means of the game theory and use Markov Decision Process to predict the most vulnerable sensor nodes [1]. It formulates the attack-defense problem as a two-player, nonzero-sum, non-cooperative game between an attacker and a sensor network. It shows that this game achieves Nash equilibrium and thus leads to a defense strategy for the network. Then, it uses Markov Decision Process to predict the most vulnerable sensor node. Finally, it uses an intuitive metre (nodes traffic) and protects the node with the highest value of this metre. Indeed, wireless sensor networks are susceptible to many forms of intrusion. In wired networks, traffic and computation are typically monitored and analyzed for anomalies at various concentration points. This is often expensive in terms of networks memory and energy-consumption, as well as its inherently limited bandwidth. Wireless sensor networks require a solution that is fully distributed and inexpensive in terms of communication, energy, and memory requirements. In order to look for anomalies, applications and typical threat models must be understood. It is particularly important to understand how cooperating adversaries might attack the system. The use of security groups may be a promising approach for decentralized intrusion detection [27]. Apart from those more general approaches, some papers provide intrusion detection techniques for particular operations. [11] describes a distributed algorithm, BoundHole, building routes around the routing holes, which are connected regions of the network with boundaries consisting of all the stuck nodes. It shows that hole-surrounding routes can be used in geographic routing, path migration, information storage mechanisms and identification of regions of interest. [9] proposes a general scheme to detect localization anomalies that are caused by adversaries. It formulates the problem as an anomaly intrusion detection problem, and proposes a number of ways to detect localization anomalies. [24] describes an intrusion detection technique that uses information on both the network topology and the positioning of sensors to determine what can be considered malicious in a particular network place. The technique relies on an algorithm that automatically generates the appropriate sensor signatures. It applies that approach to an intra-domain distance-vector protocol and reports the results of its evaluation. Moreover, there are some papers applying fault-tolerant technologies in providing network security. In [4], secure multi-path routing to multiple destination base stations is designed to provide intrusion tolerance against isolation of a base station. Also, anti-traffic analysis strategies are proposed to help disguise the location of the base station from eavesdroppers. [7] targets the identification of faulty sensors and detection of the events reachability in sensor networks with faulty sensors. It proposed two algorithms for faulty sensor identification and fault-tolerant event boundary detection. These algorithms are localized and scaleable for WSNs. In [29] the authors propose a statistical approach in order to detect attacks. Each cluster is equipped with a node that contains the rules in order to perform the detection. However, this statistical solutions have a high false alarm rate hence wasting energy in order to deliver the attacks. Moreover, a single sensor that performs all the detection activities will consume its energy before other sensors and the monitoring activities will soon be deactivated. Our approach aims to distribute the intrusion detection activities in order to prolong the monitoring. In [26] an intrusion detection scheme based on extended finite state machines is presented. They need to model one automaton for each link of the network in order to specify the correct behavior among all the connections of ad hoc networks. A backward checking algorithm detects run-time violations of this specification. With respect to it, our approach for WSNs, massively divide the workload of the monitoring activity among all the sensors participating in the given protocol. Moreover, as input, we do not need to model each link of the network but just a Global Automaton that specifies the behavior of the given protocol. Then, such an automaton is automatically divided and distributed among sensors according to their actual roles in the network.

1.2 Outline

The paper is organized as follows. In the next section we introduce some notation coming directly from the wired environment where IDSs usually are successfully applied. We point out basic difference with the wireless and mobile environment that must be considered for managing WSNs. In Section 3, the CoP protocol is introduced. We make use of it in order to better explain our approach at work. CoP is in fact a protocol designed for routing on mobile WSNs. Our aim, while describing our method, is to enrich the protocol by means of security issues. Section 4 gives details about the formalization needed to describe the system interactions defined by the CoP protocol while Section 5 makes explicit the state machine needed as input by our tool in order to describe the desired behavior of the sensor network. We show how our tool can be tuned in order to output different levels of security. Section 6 describes the heavy solution where all the sensors of specific areas monitor each other. In section 7, instead, we show how the same kind of monitoring can be performed with the minimum overhead by means of a subdivision of the monitoring according to the

sensor roles. Section 8 is devoted to experimental results. It shows the overhead obtained on the CoP protocol. Finally, Section 9 provides conclusive remarks.

2 From the wired to the wireless environment

In this section we introduce some basic definitions and the notation that come directly from the wired and static environment. As described in the introduction, this is due to the fact that intrusion detection systems are usually more suitable for those kinds of networks. Next, we point out the main properties that must be considered when moving to a mobile and wireless environment.

A wired system S is composed by a set of entities $\{C_1, \ldots, C_n\}$, communicating by means of the $send(sender,\ receiver,\ msg)$ and $receive(sender',\ receiver',\ msg')$ primitives. $sender$, $receiver$, $sender'$ and $receiver'$ are formal parameters that take as value a component in S. msg and msg' are formal parameters that take values in the domain of all possible entities messages values. The $send(C,\ C',\ m)$ ($receive(C,\ C',\ m')$) primitive invocation is used by an entity C (C') in order to send (receive) the message m (m') to (from) the entity C' (C). A communication is then achieved whenever a $send$ invocation is eventually followed by the related $receive$ invocation. In the following we assume that a global system clock exists. However, this assumption is needed only for modeling purposes and it will be relaxed in Section 7.

Given a system S, the $send$ and $receive$ primitives invocations define an architectural system trace, i.e., a string containing all $send$/$receive$ invocations performed by the entities of the system.

Definition 1 *Let $S = \{C_1, \ldots, C_n\}$ be a system. Let T be a string $p_1 p_2 \ldots p_{i+1} \ldots$. T is an* architectural system trace *of S if the following properties hold: (i) $\forall k \geq 0$, p_k is either a $send$ or a $receive$ invocation performed by an entity C of S; (ii) if $i < k$ then $t(p_i) \leq t(p_k)$, where $t(p)$ denotes the global system time at which the invocation occurs.*

The primitives invocations performed by an entity produce an entity local trace, i.e., all invocations locally observed on a single entity:

Definition 2 *Let $S = \{C_1, \ldots C_n\}$ be a system. Let C be an entity of S. $T_C = p_1^c\ p_2^c\ \ldots\ p_k^c p_{k+1}^c\ \ldots$ is an* entity local trace *of C if $\forall k \geq 0$, p_k^c is either a $send$ or $receive$ invocation performed by C.*

In the following we define the merge operation among entity local traces.

Definition 3 *Let $S = \{C_1, C_2, \ldots C_n\}$ be a system. Let C_1 and C_2 be two entities of S. Let $T_{C_1} = p_1^{c_1}\ p_2^{c_1} p_3^{c_1}\ \ldots\ p_k^{c_1} p_{k+1}^{c_1} \ldots$ and $T_{C_2} = p_1^{c_2}\ p_2^{c_2} p_3^{c_2}\ \ldots\ p_j^{c_2} p_{j+1}^{c_2} \ldots$ be the local entity traces of C_1 and C_2, respectively. A* merge trace *$T_{C_1} \oplus T_{C_2}$ is a new trace defined by $p_1\ p_2 p_3 \ldots p_t p_{t+1} \ldots$ where: (i) p_t appears in $T_{C_1} \oplus T_{C_2}$ if and only if p_t appears either in T_{C_1} or T_{C_2}; (ii) for each p_i and p_j appearing either in T_{C_1} or T_{C_2} if $t(p_i) < t(p_j)$ then p_i appears before p_j in $T_{C_1} \oplus T_{C_2}$.*

In our system model each running entity C_i defines an entity local trace T_{C_i} and the merge of all entity local traces define an architectural system trace T.

For the ease of notation we denote the invocation $send(\ C_1,\ C_2,\ m_1)$ ($receive(\ C_3,\ C_4,\ m_2)$) with $!m_1\_C_1\_C_2$ ($?m_2\_C_3\_C_4$), i.e., we substitute $send$ ($receive$) with the symbol ! (?) followed by the message m_1 (m_2) and the sender (receiver). Moreover, we denote with p^C an invocation performed by C that can be either of the form $!m_1\_C\_C_1$ or $?m_2\_C_2\_C$.

This system model, characterized by $send$ and $receive$ invocations, has been shown to be flexible enough to model different distributed systems interactions and communication patterns. Messages can be structured as needed for security and modeling purposes (e.g. a message can be a service invocation, hardware signals, control signals).

Generally speaking, we assume that each entity C implements an interface. The C interface contains each C required and provided service (if any) described using the notation $!serviceName(listOfParameters).returnType$ and $?serviceName1(listOfParameters1).returnType1$, respectively. The former means that C is a client of the $serviceName$ service having the $listOfParameters$ formal parameters and the $returnType$ returned value type. The latter means that C is a server of the $serviceName1$ service. The $listOfParameters$ is a sequence of the type $T_1\ D_1\ X_1$, $\ldots$, $T_n\ D_n\ X_n$, i.e., a sequence of parameter declarations. A parameter declaration $T_i\ D_i\ X_i$ can be described as follows. T_i is a label that can take one of the following values: in and out. in specifies that the parameter X_i is an input service parameter (i.e., it must be provided to the service). out specifies that X_i is an output service parameter (i.e., it contains an output after the service execution). D_i is the domain of X_i and defines the set of values that it can take. Finally, a service $returnType$ is the set of values that the service can return to the environment.

The entities then communicate by means of services invocation. There are three main types of entities communication models [3]: (i) synchronous; (ii) asynchronous; (iii) deferred synchronous. As it is implemented in the CORBA interface definition language the type of service invocation is specified by means of a label that is prefixed to the service declaration. However for the sake of brevity we do not show such detail in all entities interfaces described in this paper.

In the following we show how this architectural model can be mapped to our system one. Let C (C') be an entity that requires (exports) the service ! (?) $serviceName$ (T_1 D_1 X_1, ..., T_n D_n X_n).$returnType$. Synchronous and deferred synchronous service invocations from the client C to the server C' are modeled with the following sequence of $send$ and $receive$ invocations: (1) $!m\_C\_C'$ where $m = serviceName(x_1, \ldots, x_n)$, and $x_i \in D_i$, (i.e., the client invocation); (2) $?m\_C\_C'$ (i.e., the server incoming request); (3) $!m'\_C'\_C$ where $m' = serviceName.$ and $serviceName.$ encodes the C' outgoing response (i.e., the server answer); (4) $?m'\_C'\_C$ (i.e., the client incoming result). An asynchronous service invocation is modeled as above without steps 3 and 4.

In this case the structure of messages sent by means of our primitives is strictly related to the entity interfaces. If C requires the service $!serviceName$ ($T_1 D_1$ X_1 , ... , T_n D_n X_n).$returnType$ then C can send a message $serviceName(X_1, \ldots, X_n)$, where X_1, ... X_n are suitably instantiated and can receive the incoming message $serviceName.$ (i.e., the service invocation result) belonging to the set $returnType.$ Symmetrically, If C' provides the service $?serviceName1$ ($T_1 D_1$ X_1 , ... , T_n D_n X_n). $returnType1$ then C' can receive a message $serviceName1(X_1, \ldots, X_n)$, where X_1, ... X_n are suitably instantiated and can send the outgoing message $serviceName1.$ (i.e., the service invocation result) belonging to the set $returnType1.$ Therefore from now on messages can have the above structure and are suitably instantiated when they are sent.

Note that the above system model implicitly assumes that: (i) both sender and receiver of a message are known; (ii) an entity cannot change its location; (iii) an entity cannot change its interface and behavior. In the process of adapting the previous model to mobile WSNs we match entities with sensors and entity interfaces with roles. Clearly the previous assumptions cannot be now guaranteed. The more evident is (ii) since we are in a mobile environment. For (i), due to the random and mobile environment, a sensor cannot know its neighbors at each time unless it wastes energy in control messages. Concerning (iii), in general, in WSNs sensors change their behavior according to their actual role in the network, i.e., a sensor can change the messages it can send/receive. The idea is usually to guarantee a balanced energy-consumption of all the sensors participating in the protocol. As we are going to see in the next section, in the CoP protocol [23] the roles change depending on the actual positions of the sensors. We therefore remove assumptions (i), (ii) and (iii), hence obtaining a framework able to work with all those protocols for WSNs in which sensors have different behaviors according to a predetermined set of roles. The security aspects that we want to guarantee by this method concern the monitoring of the normal behavior

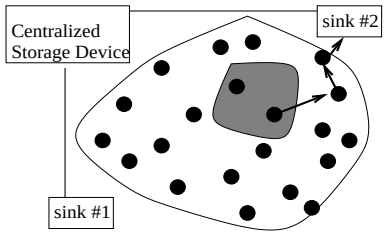

Figure 2. Multihop routing from the area of interest (shaded area) to the sink in a sensors field.

of the sensors participating in a given protocol. In order to better describe and validate our new approach, we apply it on a protocol designed for mobile WSNs, CoP [23]. Before giving the details of our approach according to the CoP protocol, in the next section we first present how it routes messages on random instances of uniform sensors distribution.

3 Connectionless Probabilistic (CoP) routing

The CoP protocol [23] for routing on mobile WSNs assumes a random uniform distribution of sensors inside a given area. The only thing that each sensor needs to know in order to participate in the CoP protocol is its own location, the location of the sink (the fixed infrastructure outside the sensed area (see Figure 2)) and a further parameter called *grid unit*.

In such a model, a communication session begins when a sensor needs to inform the sink about some collected information of interest. Since computing operations is cheaper than transmissions (see for instance [12, 22]), aggregating information is desired. As in the great majority of the protocols that are proposed, the aim is to efficiently route the message towards the destination, using the least possible energy in order to extend the lifetime of the whole network. To this end, the standard multihop approach was adopted here as well. Location-awareness routing protocols for ad-hoc networks typically assume some kind of awareness of a greater topology among the distributed sensors. Quite usually this means that in order to make local decisions, the nodes are required to know their neighbors' positions as well as their own. This is achieved by exchanging control messages that consume considerable amounts of energy in large, densely deployed, mobile networks. The key idea of the CoP protocol is that the saving of energy is achieved not only by choosing an appropriate path between source and destination pairs but also by eliminating all the transmissions usually needed by other protocols to choose the next hop node or just to communicate the positions of the nodes

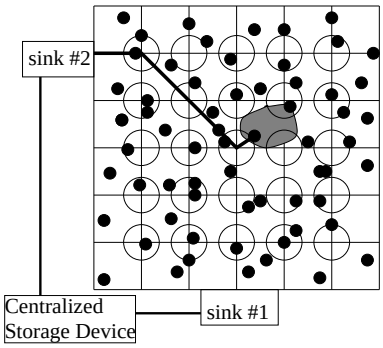

Figure 3. Multihop routing from the area of interest (shaded area) to the sink in a sensors field using the virtual grid. The empty circles represent the area associated to each virtual grid node. Every node inside such an area participate in a local leader election in order to become clusterhead.

(see for instance [2, 18, 20]). Furthermore, since we assume mobility in our model, the determination of static paths or the knowledge of the neighbors' locations could be useless in many cases where real-time connectionless communication is required.

In Cop, clustering methods are used to reduce the number of needed hops to establish the required communication session and hence reduce the average routing time. To this end, a two-level communication model was proposed where each node is itself candidate to be either a normal sensor or a clusterhead.

The protocol assumes a virtual infrastructure represented by a grid covering the sensed area. Messages are then routed on the grid nodes emulated by the sensors. The knowledge of the grid is determined by considering the sink as a grid node at the border of the area of interest and according to the fixed grid unit. Since the sensors are randomly spread on the area of interest, a distortion parameter called ds is fixed to be the maximum distance from a virtual grid node (VGN) where the real sensor has to reside in order to candidate itself and become a clusterhead (see Figure 3). Roughly speaking, this means that all the sensors in the fixed range of a VGN "believe" they are grid nodes. All the other remaining sensors are themselves associated to some VGN just through the minimum distance, hence determining an area (A_{VGN}) associated to each VGN. Note that if more than one sensor resides inside a described circular area, a standard local "leader election" is performed [21] in order to elect the clusterhead. In [23] it is formally described how to estimate the right value for ds in order to fix with high probability that at least one sensor resides inside each A_{VGN}. The configuration can easily change with time, according to the degree of the sensors' mobility but each one can decide which is the closest clusterhead-area or if it is a clusterhead itself. If a sensor is a clusterhead, it can transmit the collected information to the next clusterhead-area in order to reach the sink. Since the transmission needed power of a non-clusterhead node is less than the one needed by a clusterhead, in order to prolong the lifetime of the entire network, a sort of rotation in the roles could be convenient. If the network is characterized by high mobility, then every node frequently changes its status from clusterhead to non-clusterhead and vice-versa. This depends on the actual location of the sensor and therefore mobility works in favor of a fair and uniform energy consumption in the CoP protocol.

4 Adapting the model

In the field of location-awareness and clustering protocols like CoP, we model a mobile WSN by a set of sensors $AH = \{s_1, s_2, \ldots, s_k\}$. Let $L \subseteq AH$ be a subset $\{l_1, l_2, \ldots, l_m\}$ of sensors identifying the clusterhead of a given protocol P. In other words, the sensors in L characterize a set of areas $Ar = \{Ar_1, Ar_2, \ldots, Ar_m\}$ (clusters) where each area Ar_i represents the portion of the sensed area where the corresponding l_i plays the clusterhead role. There can be different roles according to P, let $R = \{C_1, C_2, \ldots, C_n\}$ be the set of roles. Each C_i has associated an interface that characterizes all messages sent/received by sensors playing that role.

In wireless sensor network we refine the semantic of invocation. An invocation of the form $!m\_C\_C_1$ means that a sensor playing the role C inside some area $Ar_i \in A_r$ sends the message m to all sensors residing in the same area Ar_i playing the role C_1. An invocation of the form $?m_1\_C\_C_3$ means that all sensors playing the role C_3 inside some area $Ar_i \in A_r$ receive the message m_1 sent by a sensor playing the role C inside Ar_i. Due to the open medium access, in the system model for WSNs each sensor residing in a given area belonging to A_r can sniff all the traffic occurring inside such an area. In this model each send invocation is then immediately followed by the related receive. Therefore, in our policies, it is redundant to model the reception of a message that is assumed to follow the related send.

In some invocation we can use the symbol '*' instead of a role C whenever the invocation is performed by an unknown role. We can associate to each role a set of variables that are used in the policies definition. Moreover, we specify when a role is played by exactly one sensor or by different sensors.[2]

In general a sensor s can change its role from C to C' as a consequence of: (i) changing the location; (ii) the messages

[2]This specification is associated with the role definition, but in this paper we omit such detail.

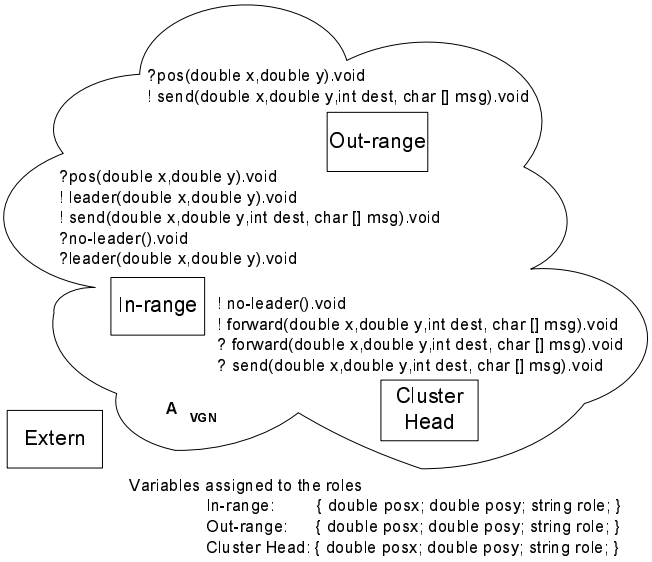

Figure 4. The CoP roles

locally sent/received by s.

According to the previous definitions, we can recognize one architectural system trace (see Definition 1) for each defined area A_{r_i}. Each running sensor s_i of the area A_{r_i} has a role R_j and defines an entity local trace T_{C_i}. The merge of such local traces generates the architectural system trace of A_{r_i}.

In Figure 4 we show the four types of roles that each sensor can play and the corresponding interfaces according to the described CoP protocol. Considering each A_{VGN} we define the Out-range, the In-range and the Clusterhead roles that are played by sensors residing in it, and the Extern role representing the sensor playing the clusterhead role inside an adjacent A_{VGN}.

The role Out-range models a sensor s located in a position that is inside the A_{VGN} but at a distance greater than ds from the corresponding VGN. This role defines an interface composed by the $?pos(double\ x, double\ y).void$ and $!send(double\ x, double\ y, int\ dest,\ char\ []\ msg).void$ asynchronous services. $?pos(double\ x, double\ y).void$ specifies that the sensor s can receive the incoming message $pos(double\ x, double\ y)$ used to set up its initial position. The parameters $double\ x$ and $double\ y$ are suitably instantiated with the coordinates of s inside the virtual grid. $!send(double\ x, double\ y, int\ dest,\ char\ []\ msg).void$ specifies that s can send the message msg towards the sink $dest$. The parameters x and y are instantiated with the current position of s and $dest$ is an integer that denotes a sink.

The role In-range models a sensor s located in a position inside an A_{VGN} and at a distance at most ds from the corresponding VGN. This role adds to the Out-range role the following services: $!leader(double\ x, double\ y).void$, $?no - leader().void$ and $?leader(double\ x, double\ y).void$. The service $!leader(double\ x, double\ y).void$ specifies that the sensor s can send the message $leader(double\ x, double\ y)$ in order to become clusterhead. The parameters $double\ x$ and $double\ y$ are suitably instantiated with the coordinates of s inside the virtual grid. $?no-leader().void$ is implemented by s in order to accept the notification sent by the clusterhead when it leaves its role. $?leader(double\ x, double\ y).void$ is used by s to receive the notification of a sensor s' that requires to be clusterhead. The parameters $double\ x$ and $double\ y$ are suitably instantiated with the coordinates of s'.

The role Clusterhead is played by a sensor s providing the forward of messages towards the right sink. This role defines the following services: $!no - leader().void$, $?forward(double\ x, double\ y, int\ dest, char\ []\ msg).void$, $!forward(double\ x, double\ y, int\ dest, char\ []\ msg).void$ and $?send(double\ x, double\ y, int\ dest, char\ []\ msg).void$. The $!no - leader().void$ service specifies that the sensor s can send the asynchronous message $no-leader()$ to the environment. This message is sent by s in order to leave its clusterhead role due to its movement or to its draining battery. $?forward(double\ x, double\ y, int\ dest, char\ []\ msg).void$ implements the service used by s in order to receive the message msg. This message is forwarded by a clusterhead s' that resides in an area surrounding the one of s. The parameters x and y denotes the position of the clusterhead s' and $dest$ encodes the sink. The service $!forward(double\ x, double\ y, int\ dest, char\ []\ msg).void$ is used by s in order to forward the message msg towards the sink $dest$. The parameters x and y denotes the position of s. The service $?send(double\ x, doubley, int\ dest, char\ []\ msg).void$ is used by s in order to receive a message msg sent by a sensor s' residing inside the A_{VGN}. The parameters x and y denote the position of s' and $dest$ is an integer that denotes a sink.

The role Extern models one of the clusterheads surrounding the current A_{VGN}.

All roles have associated the real numbers x and y used to store the current position of the sensor and the string $role$ that encodes the current role played by the sensor.

5 The formal specification: Global Automaton

We use a Global Automaton to define the correct interactions among sensors (i.e., the correct architectural system traces).

Definition 4 *A Global Automaton is* 5*-tuple* $A = (Q, q_0, I, \delta, P)$ *where: (i)* Q *is a finite set of automaton*

states; (ii) $q_0 \in Q$ is the initial state; (iii) I is a finite set of input symbols; (iii) $\delta : Q \times I \rightarrow Q$ is a transition function; (iv) $P = \{P_1, P_2, \ldots, P_n\}$ is a finite set of predicates.

We use integers to denote state names, in particular 0 will be always used to denote the initial state q_0. I is a finite set of invocations. δ represents the policy that defines the correct sequence of invocations (i.e., the correct exchange of messages among sensors). Each transition rule $q' = \delta(q, p)$, with $q, q' \in Q$, and $p \in I$, has associated exactly one predicate in P denoted with $P(q, q')(p)$.[3] $P(q, q')(p)$ (e.g. $p = !m\_C\_C'$) is a boolean expression evaluated on the basis of the states q and q' and the message m. $P(q, q')(!m\_C\_C')$ acts like a transition precondition, i.e., when it is true the related transition is allowed, otherwise the transition is not permitted.

In the following we introduce definitions related to our Global Automaton, i.e., the invocations acceptance criterion and the language it recognizes.

Definition 5 *A Global Automaton $A = (Q, q_0, I, \delta, P)$ parses an architectural system trace $T = p_1, \ldots, p_i, \ldots$ one symbol at a time from left to right. Let $q_{i-1} \in Q$ be the current state of A and let $p_i \in I$ be the next symbol to read. A* accepts *p_i if there exists a transition rule $q_i = \delta(q_{i-1}, p_i)$ defined in A and $P(q_{i-1}, q_i)(p_i)$ is true. In this case we say that the symbol p_i can be accepted by means of the A-rule $q_i = \delta(q_{i-1}, p_i)$.*

Definition 6 *Let $A = (Q, q_0, I, \delta, P)$ be a Global Automaton and $T = p_1, \ldots, p_i, \ldots$ be an architectural system trace. Let q_0 be the starting state of A and p_1 be the first symbol to read. A* accepts the sequence *T if for each current state q_{i-1} and next symbol p_i, A accepts p_i. $q_i = \delta(q_{i-1}, p_i)$ is the new state of A and p_{i+1} the next symbol to read.*

Definition 7 *The language $\ell(A)$ recognized by $A = (Q, q_0, I, \delta, P)$ is composed of all traces accepted by A.*

We point out that this acceptance criterion permits to recognize finite and infinite sequences of symbols.

We can annotate a rule (e.g. $q_i = \delta(q_{i-1}, !m\_C\_C')$) with a piece of code composed of a sequence of instructions.[4] We use instruction in order to set the role variables.[5] Variables store the current sensor position and the role played by it. They can be updated on the basis of:(i) the m message; (ii) the role C; (iii) the current state q.

A Global Automaton models the *correct* exchange of messages among sensors playing different roles and residing in (i) the same area; or (ii) an adjacent area. Predicates can be used to define the correct messages format so that some kind of attacks are avoided (e.g. sql injection, buffer overflow).

5.1 The Global Automaton definition for the CoP protocol

Starting from the description of the CoP protocol we now point out some basic properties that should be guaranteed in order to obtain a fair behavior of the protocol.

1. For each grid node there must be at most one sensor playing as clusterhead.
2. When a finite amount of data has been collected by a clusterhead, it must be forwarded in the correct direction.
3. A clusterhead that changes its status to normal sensor due to a movement or because of the draining battery has to forward all the collected messages before its movement.
4. All messages forwarded by a clusterhead have to be received by the clusterhead of the adjacent VGN area.
5. When a clusterhead leaves its role a new sensor (if any in the area) has to take its role.

We formalize these properties by defining a state machine that will be given in input to our tool in order to produce the distributed "patch" for the sensors participating in the CoP protocol.

Figure 5 shows the Global Automaton related to the sensors based system of Figure 4. This automaton defines the correct sequences of messages inside each A_{VGN}. At the beginning (state $q0$) all the sensors are informed about their positions. As described in Section 3, according to their position, each sensor sets its local variable $role$. The In-Range sensors candidate themselves to become leader. Once the ClusterHead has been elected, the system move to state $q1$ and the real interaction can start. This transition, in practice, realizes property 1. Property 2 is realized by the path $q1, q2, q3$. In this example we fixed the "finite amount of data" by means of a maximum of three collected messages which the clusterhead necessarily forwarded. Property 3 is realized by means of transition $q1, q5$, in fact, if the system state is $q1$ there are no messages stored in the clusterhead. When data is forwarded, it is received by the Extern role, i.e., some clusterhead of another A_{VGN} on the way to the specified sink.[6] And this realizes property 4. Finally, prop-

[3] For the sake of brevity we do not show the syntax and the semantic of the predicates. These implementation details are described in [25] where we show as each predicate is constituted by atomic formulae connected by means of *and*, *or* and *not* operators. Atomic formulae involve message parameters, constants and the usual operator $==, != , >$, $sizeof$ and so on.

[4] The instructions are specified in the same language used to implement the IDS.

[5] Each sensor playing the role C has its local copy of these variables.

[6] Note that, in our example, while a clusterhead is collecting messages (i.e., the system is either in $q2$ or $q3$ or $q4$), it is not allowed to receive a forward. This, in fact, can happen only at $q1$. In order to not waste messages, this means that, according to the scheduling at the MAC layer, there is some time that is a priori set up. During such a time a clusterhead can wait for other messages without incurring in any forward.

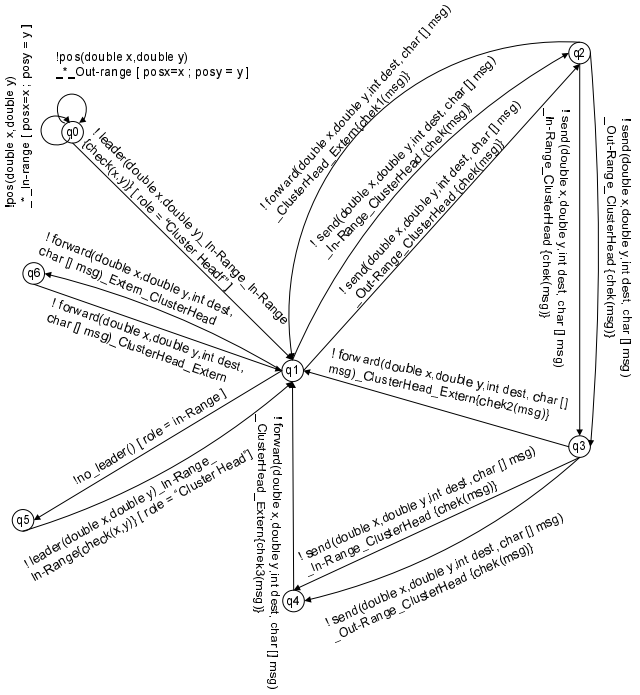

Figure 5. Global Automaton

erty 5 is valid by means of transition $q1, q5$. From $q5$, in fact, a new ClusterHead must be elected before any other communication can occur.[7] Note that, when a ClusterHead receives a forward (transition $q1, q6$), it necessarily has to forward it (transition $q5, q1$).

Concerning the predicates, $check1(msg)$ is used to verify the correct format of the message msg forwarded by the ClusterHead. This predicate permits to check that msg is not a buffer overflow attack. The $check2(msg)$ is similar to the above one, however it adds a test verifying that msg is equivalent to the compression of the two messages previously received by the ClusterHead. The predicate $check(x, y)$ verifies that the leader is at a distance at most ds from its VGN.

We use the Global Automaton as the basis to build our specification-based intrusion detection system. We produce a set of filters (one for each specified role) that will be assigned to each sensor according to their actual role. Each filter locally monitors the sensor it resides on in order to validate the policy expressed by the Global Automaton. We provide different solutions in order to implement the filters:

1. a 'heavy' solution in which each filter contains the whole global automaton;

2. a 'light' solution in which each filter contains parts of the Global Automaton (in the following referred to as local automaton). These parts are the minimum required in order to locally simulate the Global Automaton.

3. 'intermediate' solutions in which the local automaton produced in 2 is enriched with further Global Automaton parts.

These solutions allow a tradeoff among the security level and the overhead generated by the filter addition.

6 Heavy solution

In figure 6 we describe the heavy solution for a filter that resides on a sensor s playing the role C.

At step 1 the filter observes an invocation $!m\_C_1\_C_2$. At step 2 the filter takes into account the interface of the role C_1 and checks whether m can be derived from such an interface. When the derivation of m fails an attack reaction policy is undertaken.

At step 3 the filter verifies whether the current automaton state (e.g., q) contains a rule $q' = \delta(q, !m\_C_1\_C_2)$ such that the predicate evaluated on $!m\_C_1\_C_2$ is true. In this case the filter applies one of the following steps 3a, 3b, 3c or 3d.

Step 3a is applied when the invocation $!m\_C_1\_C_2$ is performed by s that is playing the role C, with $C \equiv C_1$. In this case the filter performs the invocation (i.e., it forwards the message m to the environment), changes the automaton state (that becomes q') and executes the code related to the rule $q' = \delta(q, !m\_C_1\_C_2)$. Step 3b is applied when the filter sniffs $!m\_C_1\_C_2$ from the environment. The filter changes the current automaton state to q' and in the case that s is playing the role C, with $C \equiv C_2$, the message m is forwarded to s. Note that, a receive invocation from the filter point of view immediately follows the related send invocation. Step 3c is applied when the filter detects that s has performed a send invocation $!m\_C_1\_C_2$ and s is not playing the role C_1. In this case the message is discarded and a reaction policy is undertaken. Symmetrically, step 3d is applied when $!m\_C_1\_C_2$ is sniffed from the network and it was not performed by a filter s' playing the role C_1. At step 4 the filter cannot accept the invocation in the current state, thus the invocation is discarded and an attack reaction policy is undertaken. Note that the interfaces are used in order to: (i) verify the correct format of a message; (ii) establish when the filter has to pass the message to its sensor (i.e., the role defines an incoming service).

The reaction policy could require the sending of an alert message. This kind of message contains the information of the violation. The following reaction rule are applied: (i) if the attack derives from a sensor invocation (i.e., it is locally detected by the related filter) then the filter locally discards the message and no alert is sent; (ii) if the filter

[7] Again, in order to not waste forward messages we may think of a buffer for the In-Range roles in which a forward is temporarily stored till a new ClusterHead is elected.

1. The filter observes an invocation $!m\_C_1\_C_2$, if any.
2. If m cannot be derived from the interface of the role C_1 then the invocation is discarded and a reaction policy is undertaken: go to step 1.
3. If in the current automaton state (e.g., q) the filter can accept the invocation $!m\_C_1\_C_2$, i.e., there exists a rule $q' = \delta(q, !m\_C_1\_C_2)$ and the predicate $P(q, q')(!m\_C_1\_C_2)$ is true, then
   (a) if s is playing the role C, with $C \equiv C_1$, and the filter receives the invocation $!m\_C_1\_C_2$ from s then
      i. the invocation is performed by the filter of s, (i.e., the message m is forwarded to the environment)
      ii. the current automaton state is changed to q'
      iii. the code related to the rule $q' = \delta(q, !m\_C_1\_C_2)$ is executed: go to step 1.
   (b) if the filter sniffs $!m\_C_1\_C_2$ from the environment and the invocation was performed by a filter s' playing the role C_1, then
      i. the current automaton state is changed to q'
      ii. if s is playing the role C, with $C_2 \equiv C$, then the filter forwards the message m to s: go to step 1
   (c) if s is playing the role C, with $C \not\equiv C_1$, and the filter receives the invocation $!m\_C_1\_C_2$ from s, then the invocation is discarded and a reaction policy is undertaken: go to step 1
   (d) if the filter sniffs $!m\_C_1\_C_2$ from the environment and the invocation was not performed by a filter s' playing the role C_1, then a reaction policy is undertaken: go to step 1
4. If in the current automaton state (e.g., q) there is not a rule of the form $q' = \delta(q, !m\_C_1\_C_2)$ such that the predicate $P(q, q')(!m\_C_1\_C_2)$ is true, then an attack reaction policy is undertaken: go to step 1

Figure 6. The heavy filter behavior: a sensor s playing the role C

sniffs an invocation that constitutes an attack, the message is discarded and one of the filters (the "fastest" one) sends an alert message toward the sink. We remark that, this message is observed by all the other filters that have locally detected the same attack hence they do not have to send it again. The policy described in (ii) prevents that an anomalous sensor can flood the network with alert messages.

Concerning the mobility of the sensors we can have a sensor s that moves from an A_{VGN} to another one. Therefore, the filter of s has to discover the new correct local automaton state. In this case we provide two solutions. The first solution adds to all message m a short integer. This integer uniquely identifies a transition of the Global Automaton. By looking at this short integer the filter discovers which transition of A has been applied (i.e., it discovers the new state of its local automaton). In the second solution the filter sniffs the traffic and, by looking at the sequence of invocations, it finds the unique chain of rules that matches the sequence. The first solution requires to add overhead to the normal message, but it provides fast synchronization. The second solution does not add overhead, but it can require to sniff different invocations in order to guess the local automaton state.

Synchronization is also required when sensors can activate the *sleep mode*. In this modality sensors go idle to improve energy efficiency and wake up when required. The period of sleep mode is highly dependent on the considered protocol, therefore we cannot assume how long it will last. In our approach we require that when a sensor is turned on from the sleep mode there is a correct filter set up and the discovery of its new automaton state. In Figure 7 we show a sensor s playing the role C and the basic operations

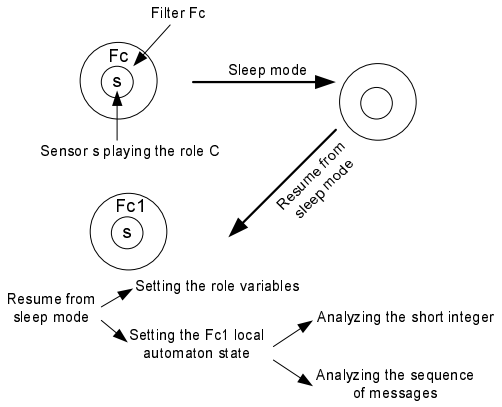

Figure 7. Sleep mode and sensor wake up

needed when s turns on from the sleep mode. The first operation updates the variable roles to set up the new sensor role (e.g., $C1$) and the related filter (e.g, F_{C1}). The second operation is the discovery of the new local automaton state. This can be done by using the solutions proposed for managing mobility, i.e., (i) read an integer added to the message in order to find out which transition of the global automaton has been applied ; (ii) sniff the sequence of invocations to find the unique chain of rules that matches the sequence.

In the following we describe how an IDS (i.e., the filter) based on the Global Automaton can be used in order to detect the attacks summarized in Section 1.

Concerning the *Compromised Node* attack, each filter controls all messages exchanged by the sensor where it resides on. Therefore, any sensor anomalous behavior is locally detected.

The *False Node* attacks is prevented since all filters residing in a given area sniff and validate all traffic. Even though new sensors (without filter) are added they must validate the property expressed by the Global Automaton since they are controlled by the 'good' ones. In particular, it is sufficient to have a good sensor, in a given area, in order to detect the compromised ones.

The *Node Outage* attack is prevented since when a sensor has to transmit something and it does not (as can happen for a ClusterHead that is not forwarding collected data), the filter of the other sensors sends an alert.

Concerning the *Message Corruption* attack, step 2 verifies the integrity of a message with respect to the defined component interface. In other words, given an invocation $!serviceName1\ (T_1 D_1\ X_1\ , \ldots , T_n\ D_n\ X_n)\_C_1\_C_2$, the role C_1 must define the service $serviceName1\ (T_1 D_1\ X_1\ , \ldots , T_n\ D_n\ X_n).\ returnType1$. Moreover, step 3c controls that the role C_2 allows the reception of the message by defining the service $?serviceName1\ (T_1 D_1\ X_1\ , \ldots , T_n\ D_n\ X_n).\ returnType1$.

Denial of Service is prevented by specifying a suitable property. Assume that from a state q to a state q' a finite number of messages must be received. Any sensor that performs an unbounded number of invocations is detected.

7 Light solution

In this section we describe the light solution in which filters do not contain the whole Global Automaton. We use an algorithm to decompose the Global Automaton in a set of local automata that are assigned one for each role. The local automaton (denoted as A_C) of the role C constitutes the basis in realizing a filter (denoted by $\Im_C$). The filter $\Im_C$ has the same behavior of the one described in Section 6, but it contains a smaller automaton. This solution preserves the correctness with respect to the heavy one and brings the advantage of predicate workload distribution. As we describe in Section 6 another method for saving energy is setting the nodes to go idle (into *sleep mode*) if they are not needed and to wake up when required. In this case the synchronization problem (see Section 6) still arises once a sensor turns on its transceiver. In Subsection 7.3 we show how to address this problem.

In the following we describe the two main phases of the algorithm, i.e., local automata generation and dependencies generation.

7.1 Local automata generation

In this phase the transitions of the Global Automaton are projected on local automata according to the described roles. The phase considers the Global Automaton $A = (Q, q_0,\ I, \delta, P)$ and a role C. The output is a preliminary version $A_C = (Q_C, q_{C0},\ I_C, \delta_C, P_C)$ of the local automaton related to the role C. A_C is obtained by considering each rule $q_1 = \delta(q, !m\_C_1\_C_2)$, with $C_1 \equiv C$, defined in A (i.e., the send invocation performed by a filter playing the role C). This rule is reflected in the A_C-rule $q_1 = \delta_C(q, !m\_C_1\_C_2)$, the states q and q_1 are added to Q_C, $!m\_C_1\_C_2$ is added to I_C and the predicate $P(q, q_1)(!m\_C_1\_C_2)$ is added to P_C. Moreover, rules of the form $q_1 = \delta_C(q, !m\_*\_C_2)$, with $C_2 \equiv C$ are added to A_C, i.e., when the sender of a message is not modeled, the receiver must check the invocation. For both the above projections we say that the invocations are 'associated' to the role C. In other words, looking at the Global Automaton A, interactions that happen locally on a sensor playing the role C are projected on A_C.

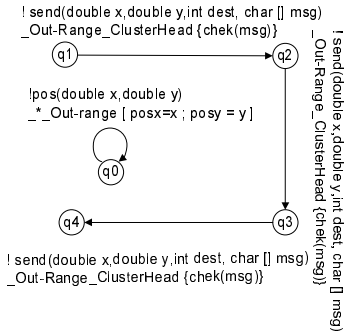

Figure 8. Projection of the Global Automaton to the Out-Range role

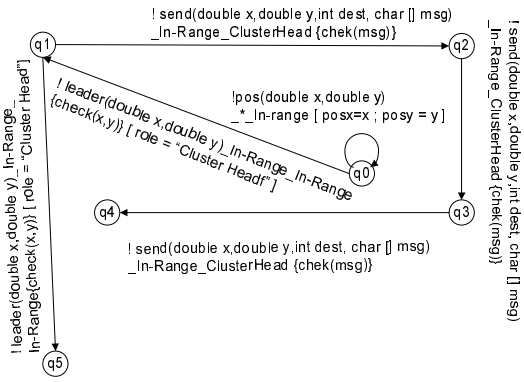

Figure 9. Projection of the Global Automaton to the In-Range role

Figures 8, 9 and 10 show the local automata generation projection performed on the In-Range, Out-Range and ClusterHead roles, respectively.[8] For instance, part of the

[8] Again, the automaton related to the Extern role is just the view of the

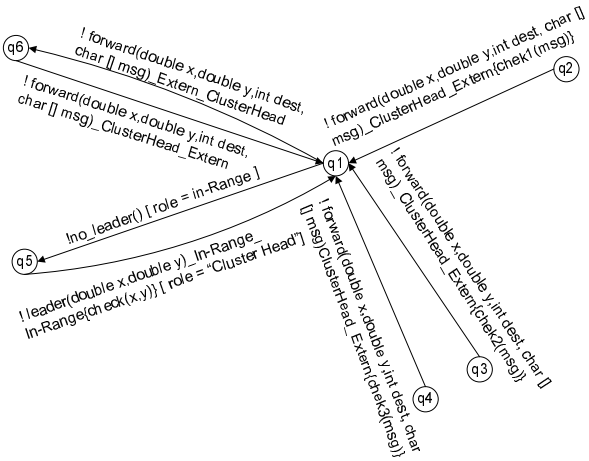

Figure 10. Projection of the Global Automaton to the ClusterHead role

Global Automaton related to the invocations: $!pos(double\ x, double\ y)\_ * \_Out - range\ [posx = x; posy = y]$, and $!send(double\ x, double\ y, int\ dest,\ char\ []\ msg)\ \_Out - Range\_ClusterHead\{chek(msg)\}$ constitute the local automaton of the role Out-Range, since such invocations are locally performed by a sensor playing that role.

The preliminary version of a local automaton is not sufficient to realize the correct monitoring. A local automaton A_C can be constituted by disconnected sub-automata (see Figure 8). The filter $\Im_C$ cannot be able to choose the right sub-automaton. Moreover, given a sub-automaton it cannot establish the next one.

Our solution is to enrich local automaton with context information and to link the sub-automata with ε-moves. We call the context information *dependency information*.

Definition 8 *Dependency information is of the form $?f(m, C')$ where C' range on the name of the sensor roles. $?f(m, C')$ means that the message m is sent by a filter playing the role C'.*

Given a local automaton $A_C = (Q_C, q_{0C}, I_C, \delta_C, P_C)$ dependencies are added to the set I_C and their correct ordering is defined by means of the function δ_C. They establish the minimal information that $\Im_C$ must sniff in order to locally validate the property expressed by the Global Automaton. Dependencies are added by the dependencies generation phase.

7.2 Dependencies generation

Dependencies affect synchronization and enabling states of the Global Automaton $A = (Q, q_0, I, \delta, P)$.

ClusterHead automaton from an adjacent A_{VGN}.

A synchronization state q of A is a state exited by at least two transitions labeled with invocations associated to different roles. We consider the case in which from q exactly two transitions exit: $q_1 = \delta(q, !m_1\_C_1\_C_2)$ and $q_2 = \delta(q, !m_2\_C_3\_C_4)$ with $C_1 \neq C_3$, and $q \neq q_1 \neq q_2$.

The local automata generation ensures that these transitions are projected onto the related local automata (i.e., $q_1 = \delta_{C_1}(q, !m_1\_C_1\_C_2)$ and $q_2 = \delta_{C_3}(q, !m_2\_C_3\_C_4)$ are added to A_{C_1} and A_{C_3}, respectively). The dependencies generation phase considers the synchronization state q and adds the rules $q_2 = \delta_{C_1}(q, ?f(m_2, C_3))$ and $q_1 = \delta_{C_3}(q, ?f(m_1, C_1))$ to A_{C_1} and A_{C_3}, respectively.

From the point of view of A, if it is in the state q then either the transition $q_1 = \delta(q, !m_1\_C_1\_C_2)$ or $q_2 = \delta(q, !m_2\_C_3\_C_4)$ can be applied. From the filters point of view such possibility is lost since these rules are independently applied by the two different filters residing on the two different sensors. The rules $q_2 = \delta_{C_1}(q, ?f(m_2, C_3))$ and $q_1 = \delta_{C_3}(q, ?f(m_1, C_1))$ are a means used by the filters to overcome this problem. Suppose that both the local automata of filters $\Im_{C_1}$ and $\Im_{C_3}$ are in the state q, if a sensor playing the role C_1 is scheduled before other ones then (i) the filter $\Im_{C_1}$ changes the local automaton by applying the rule $q_1 = \delta(q, !m_1\_C_1\_C_2)$; (ii) filters $\Im_{C_3}$ sniff the invocation $!m_1\_C_1\_C_2$ and move by applying the rule $q_1 = \delta_{C_3}(q, ?f(m_1, C_1))$. Symmetrically, if a sensor playing the role C_3 is scheduled before other sensors then (i) the filter $\Im_{C_3}$ change the local automaton by applying the rule $q_2 = \delta(q, !m_2\_C_3\_C_4)$; (ii) filters $\Im_{C_1}$ sniff the invocation $!m_2\_C_3\_C_4$ and move by applying the rule $q_2 = \delta_{C_1}(q, ?f(m_2, C_3))$.

The following definition generalizes the above considerations.

Definition 9 *Let $A = (Q, q_0, I, \delta, P)$ be a Global Automaton and let q be a state of A, we denote by $\Im^q$ a set of filters. $\Im_C$ belongs to $\Im^q$ if there exists $p \in I_C$ such that the rule $q' = \delta(q, p)$ is defined in A.*

We can observe that each local automaton A_C of the filter $\Im_C$, with $\Im_C \in \Im^q$, must contain a projection of a q-exiting transition.

If $\Im^q = \{\Im_{C_1}, \Im_{C_2}, \ldots, \Im_{C_n}\}$ and a filter $\Im_{C_i}$, with $\Im_{C_i} \in \Im^q$, and $1 \leq i \leq n$ will apply the rule $q' = \delta_{C_i}(q, !m_1\_C_i\_C')$, then all filters $\Im_{C_j}$, with $\Im_{C_j} \in \Im^q$ and $j \neq i$, move by applying the rule $q' = \delta_{C_j}(q, ?f(m_1, C_i))$. This means that all the local automaton of the filters in $\Im^q$ are synchronized to the state q. We call such dependencies *synchronization dependencies* since they are used to synchronize the filters in order to accept one invocation exactly.

A state of A is an enabling state if it is entered and exited by transitions projected on two different local automata. For instance, consider the state q in which A defines the chain of rules $q = \delta(q^-, !m_1\_C_1\_C_2)$ and $q^+ = \delta(q, !m_2\_C_3\_C_4)$,

with $C_1 \neq C_3$, and $q \neq q^-$. The local automata generation ensures that the rules $q = \delta(q^-, !m_1\_C_1\_C_2)$ and $q^+ = \delta(q, !m_2\_C_3\_C_4)$ are projected on the automata of the filters $\Im_{C_1}$ and $\Im_{C_3}$, respectively. From the A point of view, this chain of rules defines a constraint among the invocations $!m_1\_C_1\_C_2$ and $!m_2\_C_3\_C_4$. That is, $!m_1\_C_1\_C_2$ must be accepted before $!m_2\_C_3\_C_4$. However, from the local filters point of view, this constraint is lost, since the chain of rules is divided onto the filters $\Im_{C_1}$ and $\Im_{C_3}$. Therefore, the filter $\Im_{C_3}$ can autonomously perform the invocation $!m_2\_C_3\_C_4$ before that the invocation $!m_1\_C_1\_C_2$ is performed by $\Im_{C_1}$. The problem is solved by adding dependencies.

The dependencies generation adds to A_{C_3} the rule $q = \delta_{C_3}(q^-, ?f(m_1, C_1))$. Therefore, $\Im_{C_3}$ can move to the state q by means of the rule $q = \delta_{C_3}(q^-, ?f(m_1, C_1))$. However, this rule can be applied only when the filter $\Im_{C_1}$ performs the invocation $!m_1\_C_1\_C_2$, i.e., the filter $\Im_{C_3}$ sniffs the message m_1 sent by $\Im_{C_1}$. This is a mean for filter $\Im_{C_1}$ to impose the right ordering among the messages $!m_1\_C_1\_C_2$ and $!m_2\_C_3\_C_4$. We call such dependencies *enabling dependencies*, since they are used to enable the local-filter parsing when there is the right context condition.

Note that, after the addition of these dependencies, a local automaton A_C can still be disconnected, i.e., constituted by the disconnected automata A^1, A^2, ..., A^{nc}. The last step of the dependencies generation phase links together these local disconnected automata by means of ε moves. Given the state q and q_n of A_C, this step can add to A_C rules of the type $q_n = \delta_C(q, \varepsilon)$. This epsilon move models that the state q and q_n of A_C are separated by a chain of rules labeled with invocation that are not associated to the role C. The filter $\Im_C$ can ignore these interactions moving from q to q_n where it will be informed by means of a dependency when this chain of rules has been performed.

By applying usual techniques, ε-moves can be removed [13] in any local automaton $A_C = (Q_C, q_{0C}, I_C, \delta_C)$. Indeed, let $\overline{Q} = q \cup \{q_1, q_2, \ldots, q_n\}$ be the state obtained by removing ε-moves from q as explained in [13], where $\{q_1, q_2, \ldots, q_n\}$ are all the states reachable from the given state $q \in Q_C$ by means of ε-moves. The set of transitions exiting from $\overline{Q}$ can only be labeled with dependency.[9] However, the process of removing ε-moves can create a state $\overline{Q}$ where two different states $q_i, q_j \in \overline{Q}$ can be exited by transition labeled with the same dependencies $?f(m, C)$: $q' = \delta_C(q_i, ?f(m, C'))$ and $q'' = \delta_C(q_j, ?f(m, C'))$. This non determinism can be solved either by translating the local automaton $\Im_C$ to the deterministic one or by letting the filter $\Im_{C'}$ adding an integer k to the message m. The integer k uniquely identifies a transition of the Global Automaton. Therefore, although a filter can contain ε-moves only one of them can be deterministically applied.

The filter behavior $\Im_C$ is similar to the one proposed by the heavy solution: (i) it can only accept invocations associated to the role C (i.e., the ones projected to its local automaton A_C); (ii) it verifies the correct format and the predicates related to invocations performed by sensor playing the same role; (iii) it sniffs the network and moves when the message matches a dependency that labels an A_C transition. Therefore, filters based on the light solution distribute the workload of predicates evaluation. There are less rules to apply. Moreover, when the Global Automaton is big in size and a sensor never changes its role we can reduce the memory allocation.

[9] This is is shown in the on-line extended version [25].

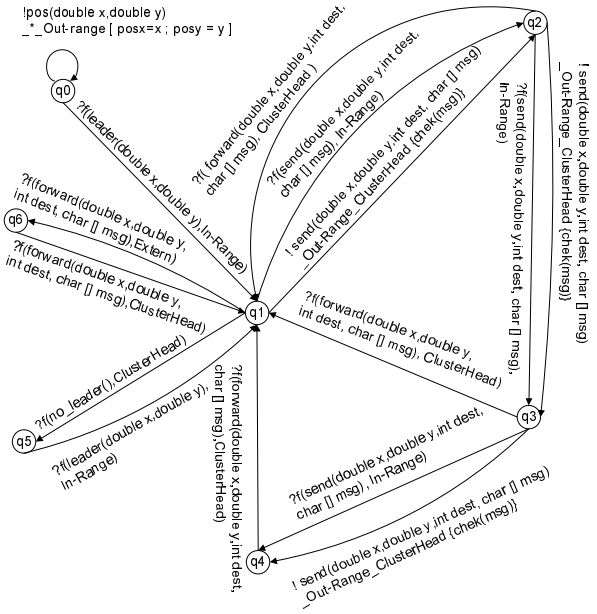

Figure 11. Local automaton of the Out-Range role

In Figure 11 we show the local automaton of the Out-Range role. We can observe as the local automaton allows the $send$ invocation only when the leader message has been sniffed from the environment. Note that only the transitions of the Out-range role require the predicates evaluation.

7.3 Mobility aspects and Sleep Mode

The position of a sensor can change as a consequence of its mobility. This may also change the role of the sensor in the network. The associated filter then, should refer to a different automaton. It can happen that a state in the original automaton is not present in the new one. In order to solve this problem we propose two solutions. One is to define a mapping function that takes in input a state q_x of a role C' and a role C'', and outputs a state q_w of the role C'' that corresponds to q_x in C'. Such a function should be evaluated by the filters each time the corresponding sensor

changes its behavior (role). The overhead is then given by the needed computations to evaluate such a function.

Another solution is to enrich the messages by making explicit the corresponding move on the local automaton, in such a way, that the receiver can understand which dependency on the local automaton is satisfied hence discovering its current state. The overhead in this case is given by the extra information we need inside each message.

The right choice among those solutions strongly depends on the given protocol. From the energy-consumption point of view the first solution seems more suitable. In the case where a sensor changes A_{VGN}, the same arguments of the heavy solution can be applied.

As already pointed out at the beginning of the section, synchronization problems can arise when the desired protocol makes use of the sleep modality for the sensors in order to save energy. In the light solution after the sensor turns on from the sleep mode its filter has to perform the same operations described in Section 6 (see Figure 7). In particular, the discovery of the new local automaton state can be done by applying the following techniques: (i) discover the new state by observing the integer added to the message in order to find out the new local state[10]; (ii) sniff the sequence of invocations to eventually find out a unique chain of rules that matches the sequence.

7.4 Attacks Detection

Concerning the attacks detection, filters behave like in the heavy solution but restricted to their own roles. The *Node Outage* attack, for instance, is prevented since when a sensor has to transmit something and it does not, the filters of the other sensors playing the same role sends an alert. Conversely, special attention must be given to *Compromised Node* attacks since we have to ensure that the Global Automaton is now perfectly emulated by the local ones. The following theorem ensures that this attack is prevented.

Theorem 1 *Given a protocol P for mobile WSNs, let $A = (Q, q_0, I, \delta, P)$ be a Global Automaton expressing the correct behaviors of p sensors $\{s_1, s_2, s_3 \dots s_p\}$ according to a set R of defined roles with respect to a cluster A_{r_i} with $1 \leq i \leq m$. Let $\Im_{C_j}$ be the filter related to the sensor s_j. The automaton A accepts an architectural trace $T = T_{s_1} \oplus \dots \oplus T_{s_p}$ if and only if all the sensor traces T_{s_j} are accepted by each corresponding $\Im_{C_j}$ with $1 \leq j \leq p$.*

Proof: $\Leftarrow$) Dependencies permit to prove that the merge of local traces is accepted by the Global Automaton.[11] For this aim we summarize the state of all filters in a given area A_{r_i} (i.e., the state of the whole IDS system) as the state of each local automaton (i.e., a p-tuple of states) and we show that the IDS state has to go through states where the following predicate is true.

Definition 10 *Let $A = (Q, q_0, I, \delta, P)$ be a Global Automaton and let $q \in Q$ be a state of A. We say that the IDS is in a state where the predicate $\psi(q)$ is true if the following conditions hold: (i) every local automaton A_C of a filter $\Im_C$ in $\Im^q$ is in the state q; (ii) every local automaton of a filter in $\overline{\Im^q}$ is in a state where all exiting transitions are labeled with incoming dependencies.*

When $\psi(q)$ is true all local automata of the filters in $\Im^q$ are in the state q, therefore exactly one filter $\Im_C \in \Im^q$ accepts its local invocation (e.g., $!m\_C\_C_1$) by means of the rule $q' = \delta_C(q, !m\_C\_C_1)$. Once $\Im_C$ has applied the rule, all other filters in $\Im^q$ move to q' by applying the rule $q' = \delta_C(q, ?f(m, C))$ (i.e., the rule labeled with a synchronization dependency). A filter $\Im_{C'}$ in $\Im^{q'} \setminus \Im^q$ defines a transition exiting from q' (e.g., $q'' = \delta_{C'}(q', !m\_C'\_C_1')$). The state q' is an enabling state of A since there is the q' entering transition $q' = \delta_C(q, !m\_C\_C_1)$ and the q' exiting transition $q'' = \delta_{C'}(q', !m\_C'\_C_1')$, with $C \not\equiv C'$, and $q \neq q'$. Therefore, if we prove that all filters in $\Im^{q'} \setminus \Im^q$ are in the state q they move to the state q' by applying the rule labeled with the enabling dependency $?f(m, C)$,[12] i.e., the IDS reaches a state where $\psi(q')$ holds. In this case we say that the IDS performs one step from a state were $\psi(q)$ is true to a state were $\psi(q')$ is true.

Starting from the initial state (i.e., all local automata are in the state q_0) the IDS performs step 1, ..., i where the predicates $\psi(q_0)$, ... , $\psi(q_i)$ are true and at each step j, with $1 \leq j \leq i$, the rule $q_{j+1} = \delta_{C_j}(q_j, !m_j\_C_j\_C_j')$ is applied by $\Im_{C_j} \in \Im^{q_j}$. The sequence of rules $q_{j+1} = \delta_{C_j}(q_j, !m_j\_C_j\_C_j')$ are projections of A rules, therefore, the Global Automaton A can accept the sequence $!m_1\_C_1\_C_1' \dots !m_i\_C_i\_C_i'$ by applying the same sequence of rules.

By induction on step i, we prove that if the filter $\Im_{C_i}$ applies the rule $q_i = \delta_{C_{i-1}}(q_{i-1}, !m_{i-1}\_C_{i-1}\_C_{i-1}')$ then all filters in $\Im^{q_i}$ are in state q_{i-1}. This is sufficient in order to prove that at step $i+1$ the predicate $\psi(q_i)$ holds.

The predicate $\psi(q_0)$ is true when the sensors start to work since all local automata are in state q_0. Moreover at step 1 if the filter $\Im_{C_1}$ applies the rule $q_1 = \delta_{C_0}(q_0, !m_0\_C_0\_C_0')$ all filters in $\Im^{q_1} \setminus \Im^{q_0}$ are in state q_0.

At step i the filter $\Im_{C_i}$ applies the rule $q_{i+1} = \delta_{C_i}(q_i, !m_i\_C_i\_C_i')$. By contradiction, we assume there exists a filter $\Im_{C'}$ in $\Im^{q_{i+1}} \setminus \Im^{q_i}$ that is not in the state q_i.

[10] It is worth noticing that this technique is the one used to manage mobility.

[11] Note that dependencies are sufficient to impose an ordering on messages; this allows us to relax the assumption that a global system clock exists (see Section 4).

[12] Note that all other filters remain in the same state since the message (i.e., the dependency $?f(m, C)$) is related to an invocation exiting from q.

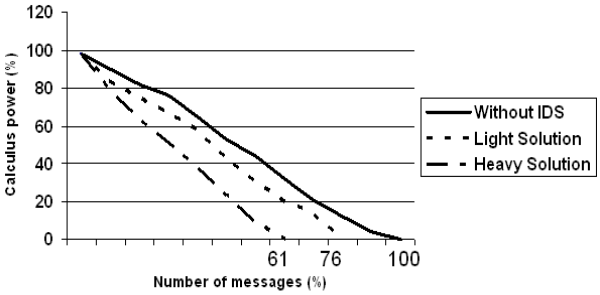

Figure 12. Average of the residual computational power depending on messages exchanged inside an A_{VGN}.

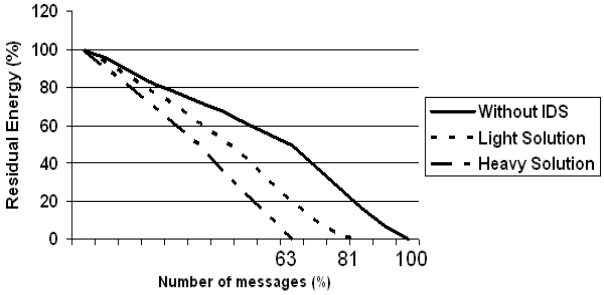

Figure 13. Average of the residual energy depending on messages exchanged inside an A_{VGN}.

The filter $\Im_{C'}$ contains a q_{i+1} exiting transition $q_{i+2} = \delta_{C'}(q_{i+1}, !m_{i+1}\_C'\_C'_{i+1})$, $\Im_{C_i}$ the rule $q_{i+1} = \delta_{C_i}(q_i, !m_i\_C_i\_C'_i)$. Therefore, q_{i+1} is an enabling state of A and $\Im_{C'}$ contains the rule $q_{i+1} = \delta_{C'}(q_i, ?f(m_i, C_i))$.

$\Im_{C'}$, at step $i-1$, cannot belong to the set $\Im^{q_{i-1}}$ otherwise by induction it would be in the state q_i at step i. Let the step j be the last state where $\Im_{C'}$ is in state q_j, i.e., for all steps k, with $k > j$, $\Im_{C'}$ does not belong to the set $\Im_{q_k}$. Thus, the chain of rules $q_{k+1} = \delta_{C'}(q_k, ?f(m_k, C_k))$ is not labeled with invocations associated to C'. With this last assumption the rule $q_{i-1} = \delta_{C'}(q_k, \varepsilon)$ is defined in $A_{C'}$. Since all rules exiting from q_{i-1} are labeled with dependency related to q_{i-1}, then $\Im_{C'}$ between step k and $i-1$ cannot move, i.e., at step $i-1$ it is in q_{i-1}, and this contradicts the hypothesis.

$\Rightarrow$)

Let $T = T_{s_1} \oplus \ldots \oplus T_{s_p}$ be a trace accepted by A. We can always find a schedule such that each sensor s_i produces the trace T_{s_i}. □

8 Performance Evaluation

In this section we show how our method affects the performance of the CoP protocol in terms of energy-consumption and computed operations. The experiments are performed running the powered protocol over hundreds of random instances of mobile WSNs. We applied both the heavy and the light solutions. We show the overhead in terms of consumed energy and in terms of performed instructions by the filtered sensors. The experiments also show the estimated percentage reduction of the network lifetime with respect to the original CoP protocol.

In Figure 12 we show the lifetime of the system inside an A_{VGN}. Considering each kind of message of the sensors as a different set of instructions, we show the overhead in terms of percentage of computational power loss. The cost of ensuring the normal behavior of the protocol in terms of number of instructions by means of the light solution is increased, on average, around 24%. The heavy solution costs, on average, around 39%. Note that transmission/reception operations are much more expensive than local computations. According to the consumption values expressed in [12, 23], transmitter and receiver electronics consume an equal amount of energy per bit, namely $5nJ/bit$. While the energy to support the signal above some acceptable threshold against power attenuation caused by the distance is just $100pJ/bit/m^2$.

In Figure 13 we show, on average, the percentage of the draining of the sensors batteries inside an A_{VGN}. For the heavy solution it is increased by around 37% while for the light solution we obtained around 19%. Note that such comparisons are performed without simulating attacks. This means that the original CoP protocol was able to perform the desired (random) communications. In the case of attacks, the original protocol can waste indeed much more energy with respect to the "safe" solutions. Consider for instance a sensor that generates traffic without following the right syntax of the messages or the right scheduling. While in the original CoP protocol those messages are forwarded till the sink, now they are just blocked by the local filters.

9 Conclusions

In the context of IDSs, we have presented a new framework able to output distributed secure protocols in the field of mobile WSNs. We started from a wired context where usually IDS features are successfully applied. We pointed out the main differences occurring when the environment is wireless and mobile and we have generated a tool that can deal with those properties. The tool takes as input the specification behavior of the correct messages exchanged according to a given protocol. It outputs a distributed IDS that guarantees the desired properties specified by means of a state machine, the Global Automaton. This automaton is

in fact split into several automata, one for each role that a sensor can play. According to the required security level, our tool can be tuned in such a way that it outputs from a light to a heavy IDS distribution. Basically the process consists in assigning to each sensor more or less control management among the traffic occurring over the network. In order to better understand the process of the IDS distribution, we have applied our tool to a recent proposed protocol for mobile WSNs, CoP [23]. We have shown how it is possible to wrap filters around the CoP sensors directly derived from the automaton given in input which describes the desired behavior of the system. The sensors are then synchronized, when needed, by means of messages exchange. The filters are then able to detect intruders locally by themselves and by means of collaborations. Finally we have observed the performance obtained by means of our tool in terms of consumed energy and computed operations with respect to the original CoP protocol. The experiments were evaluated over random instances of sensor networks by assuming no attacks occurring. Hence the comparisons were in favor of the CoP protocol but they pointed out really good performances of the safe solutions. The overhead was in fact of around 20% for the light solution up to around 40% for the heavy one. Indeed, in many cases, avoiding attacks also decreases consumed energy since it locally prevents fake traffic.

References

[1] A. Agah, S. Das, and K. Basu. Intrusion detection in sensor networks: A non-copporative game approach. In *IEEE NCA 2004*.

[2] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. In *DIALM 1999*.

[3] G. Brose, A. Vogel, and K. Duddy. *Java programming with CORBA third edition*. WILEY, 2002.

[4] J. Deng, R. Han, and S. Mishra. Intrusion tolerance and anti-traffic analysis strategies for wireless sensor networks. In *IEEE DSN 2004*.

[5] R. Di Pietro, S. Etalle, P. Havinga, Y. W. Law, and L. V. Mancini. LKHW: A Directed Diffusion-Based Secure Multicast Scheme for Wireless Sensor Networks. In *WiSPr 2003*.

[6] R. Di Pietro, L. V. Mancini, and A. Mei. Efficient and Resilient Key Discovery Based on Pseudo-Random Key Pre-Deployment. In *WMAN 2004*.

[7] M. Ding, D. Chen, K. Xing, and X. Cheng. Localized fault-tolerant event boundary detection in sensor networks. In *INFOCOM 2005*.

[8] S. Doumit and D. Agrawal. Self-organized critically and stochastic learning based intrusion detection system for wireless sensor networks. In *MILCOM 2003*.

[9] W. Du, L. Fang, and P.Ning. Lad: Localization anomaly detection for wireless sensor networks. In *IPDPS 2005*.

[10] S.-T. Eckmann, G. Vigna, and R.-A. Kemmer. Statl: An attack language for state-based intrusion detection. *Journal of Computer Security*, 10:71–104, 2002.

[11] Q. Fang, J. Gao, and L. Guibas. Locating and bypassing. routing holes in sensor networks. In *INFOCOM 2004*.

[12] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-Efficient Communication Protocols for Wireless Microsensor Networks. In *HICSS 2000*.

[13] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley publishing company, 1979.

[14] P. Inverardi and L. Mostarda. A distributed intrusion detection approach for secure software architecture. In *EWSA 2005*.

[15] P. Inverardi, L. Mostarda, and A. Navarra. Distributed idss for enhancing security in mobile wireless sensor networks. In *PCAC 2006*.

[16] P. Inverardi, L. Mostarda, M. Tivoli, and M. Autili. Automatic synthesis of distributed adaptors for component-based system. In *ASE 2005*.

[17] C. Karlof and D. Wagner. Secure Routing in Sensor Networks: Attacks and Countermeasures. In *SNPA 2003*.

[18] B. Karp and H. Kung. Greedy perimeter stateless routing (GPSR) for wireless networks. In *MobiCom 2000*.

[19] C. Ko, M. Ruschitza, and K. Levitt. Execution monitoring of security-critical programs in distribute system: A specification-based approach. *IEEE*, 1997.

[20] W.-H. Liao, J.-P. Sheu, and Y.-C. Tseng. GRID: A fully location-aware routing protocol for mobile ad hoc networks. *Telecommunication Systems*, 18(1-3), 2001.

[21] N. Malpani, J. Welch, and N. Vaidya. Leader election algorithms for mobile ad hoc networks. In *DIALM 2000*.

[22] D. Maniezzo, K. Yao, and G. Mazzini. Energetic trade-off between computing and communication resource in multimedia surveillance sensor network. In *MWCN 2002*.

[23] J. A. McCann, A. Navarra, and A. A. Papadopoulos. Connectionless Probabilistic (CoP) routing: an efficient protocol for Mobile Wireless Ad-Hoc Sensor Networks. In *IPCCC 2005*.

[24] V. Mittal and G. Vigne. Sensor-based intrusion detection for intra-domain distance-vector routing. In *ACM CSS 2002*.

[25] L. Mostarda. Distributed Intrusion Detection Systems for Secure Software Architectures. In *Ph.D. thesis, http://www.di.univaq.it/mostarda/sito/default.php 2006*.

[26] J.-M. Orset, B. Alcalde, and A. Cavalli. An EFSM-based intrusion detection system for ad hoc networks. In *ATVA 2005*.

[27] A. Perrig, J. Stankovic, and D. Wagner. Security in wireless sensor networks. *Commun. ACM*, 47(6):53–57, 2004.

[28] A. Vora and M. Nesterenko. Secure Location Verification Using Radio Broadcast. In *OPODIS 2004*.

[29] Y. Zhang, W. Lee, , and Y.-A. Huang. Intrusion detection techniques for mobile wireless networks. *Wireliless Networks*, pages 545–556, 2003.