FI MU

Faculty of Informatics
Masaryk University Brno

Improving Intrusion Detection Systems for Wireless Sensor Networks

by

Andriy Stetsko
Tobiáš Smolka
Vashek Matyáš
Martin Stehlík

FI MU Report Series **FIMU-RS-2014-01**

 March 2014

Publications in the FI MU Report Series are in general accessible via WWW:

`http://www.fi.muni.cz/reports/`

Further information can be obtained by contacting:

**Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic**

Improving Intrusion Detection Systems for Wireless Sensor Networks

Andriy Stetsko, Tobiáš Smolka, Vashek Matyáš, Martin Stehlík
Masaryk University, Brno, Czech Republic
{stetsko, xsmolka, matyas, xstehl2}@fi.muni.cz

March 31, 2014

Abstract

A considerable amount of research has been undertaken in the field of intrusion detection in wireless sensor networks. Researchers proposed a number of relevant mechanisms, and it is not an easy task to select the right ones for a given application scenario. Even when a network operator knows what mechanism to use, it remains an open issue how to configure this particular mechanism in such a way that it is efficient for the particular needs. We propose a framework that optimizes the configuration of an intrusion detection system in terms of detection accuracy and memory usage. There is a variety of scenarios, and a single set of configuration values is not optimal for all of them. Therefore, we believe, such a framework is of a great value for a network operator who needs to optimize an intrusion detection system for his particular needs, e.g., attacker model, environment, node parameters.

1 Introduction

A wireless sensor network (WSN) consists of sensor nodes – small devices equipped with sensors, microcontroller, wireless transceiver and battery. Each node monitors a physical phenomenon and sends the measurements to a base station. Since a node communication range is limited to tens of meters and it is not always feasible for the node to directly communicate with the base station, data are usually sent hop-by-hop from one node to another until they reach the base station. WSNs can support various applications for ecology and wildlife monitoring, military, building and industrial automation, energy management, agriculture, etc.

Sensor nodes are constrained in *processing power*, *memory* and mainly in *energy*. A MICAz sensor node is a typical sensor node. It is equipped with the 8 MHz Atmel Atmega128L microcontroller, 512 KB flash memory, 802.15.4 compliant Texas Instruments CC2420 transceiver and two AA batteries. The transceiver consumes 18.8 mA (with 3.3 V power supply) in the receiving mode [1], which is the most energy consuming mode. If we use two NiZn AA batteries with a nominal voltage of 1.65 V and a capacity of 1800 mAh, the estimated lifetime of a constantly receiving node is approximately 96 hours, i.e., 4 days. However, in general for WSNs one expects a functional network for the duration of time that ranges from several days to several years.

In this work, we propose a *framework that semi-automatically optimizes the configuration of an intrusion detection system* (IDS) in terms of detection accuracy and memory usage for any given scenario, e.g., a network topology, the network stack of benign sensor nodes, and anticipated attacks. We do not aim to propose particular novel techniques for intrusion detection (as such) in sensor networks and our ultimate long-term aim is to provide a framework that does not depend on a particular attacker model (or a group of models). We focus on intrusion detection since it is an essential mechanism to protect a network against internal attacks that are relatively easy and not expensive to mount in WSNs. In comparison to conventional wired and wireless networks, an attacker can often easily access the deployment area of a WSN, capture some nodes, and launch a wide range of attacks (for the list of possible attacks, see [18]).

The technical report roadmap is as follows. The conceptual architecture of our framework is described in Section 2. Section 3 contains high-level technical details of our proof-of-concept implementation. In Section 4, we describe our test case. We tested the framework using a static topology in three different scenarios – these scenarios were selected to illustrate the framework merits, Section 5 describes these scenarios and test results. We compare our approach to related work in Section 6. Finally, Section 7 concludes our work and presents plans for future work. Particular details of evolutionary algorithms that we used for optimization are then provided in Appendix.

2 Conceptual architecture of the framework

In this section, we present the conceptual architecture of our framework that semi-automatically optimizes the configuration of an IDS for a given application scenario. The framework includes an *optimization engine* and a general-purpose *network simulator*

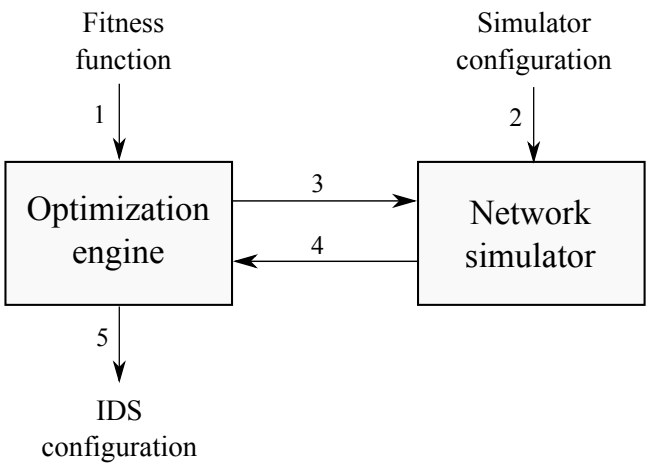


Figure 1: Conceptual architecture. The arrows depict input (output) to (from) the components of our framework.

(see Figure 1). The whole process consists of five main steps. In the first step, a network operator defines a fitness function for the evaluation of an IDS configuration. We define a reasonable fitness function in Subsection 3.3. It integrates evaluation metrics from [2] such as true positives, true negatives and memory usage. In the second step, the network operator configures the network simulator in such way that it simulates a scenario in which an IDS should be deployed. This step is described in Subsection 2.1 in more detail. The remaining three steps are completely automatic. The third and fourth steps take place in an iterative manner. The optimization engine provides a candidate configuration of an IDS to the simulator. The simulator evaluates it according to predefined metrics, e.g., detection accuracy, memory usage, and returns information required to compute the fitness function back to the optimization engine. Based on the evaluation, the optimization engine changes the values of parameters and repeats the procedure until a predefined condition holds, e.g., parameters become optimal for a given scenario, or the maximum number of iterations is exceeded. Finally, in the fifth step, the optimization engine outputs the best found (hopefully, the optimal) IDS configuration.

2.1 Simulator configuration

The network operator provides a complete network model, which among others, includes a *network topology*, *models of benign and malicious nodes*, *wireless channel and energy consumption models* (see Figure 1).

2.1.1 Network topology

The simulator provides a possibility to set a topology manually or to generate it automatically. In case the network operator knows the precise topology of the network, he can use the first option, and optimize the IDS for this particular topology. In case the topology is not known in advance, the network operator can use the second option, generate several random topologies, and optimize the IDS for all of them simultaneously. For more details, see the discussion on robustness of a found solution in Subsection 2.2.

2.1.2 Benign sensor node

Models of the node hardware and software should be provided, i.e., radio, IDS, medium access control (MAC) layer, network layer, and application layer models. There could be several types of a benign node in the network (e.g., a cluster head, a base station, a general-purpose sensor node), and they might have a different network stack. The network operator composes a benign node model from the available protocols (distributed within the simulator, or implemented by a third party). If the required protocol at a certain network layer is not available, the network operator should implement it. Further, the network operator configures the parameters of the models.

2.1.3 Malicious sensor node

Similarly, models of the node hardware and software should be provided. There could be several types of a malicious node in the network (e.g., internal/external [18], passive/active [21]). Usually, it is known to a network operator where the network will be deployed, and what the purpose of the network is. Based on this information, the network operator can estimate the risks of different attacks (e.g., selective forwarding, jamming, hello flood), and include into a simulation only those that pose a serious threat. The network operator composes a malicious node model from the modified models available within the distribution of the simulator, or implements them by himself. For example, to implement a selective forwarder, it might be enough to modify the network layer of a benign node to drop a certain percentage of incoming packets.

2.1.4 Wireless channel

The simulator can provide more than one model for radio propagation, and a network operator can choose the one that is more suitable for the environment where a network will be deployed.

2.1.5 Energy consumption

The simulator can provide more than one energy consumption model.

2.2 Discussion

In this subsection, we discuss several issues related to the framework design choices and framework usage.

2.2.1 Simulator versus testbed

We decided to use a simulator since a testbed is slow for comparison of considered alternative configurations, labour-intensive and it does not produce comparable results due to the uncontrollable factors (e.g., wireless channel effects). Candidate configurations should be tested under the same network conditions, otherwise they cannot be compared. In a testbed, we are not able to *reproduce* the same environment each time a candidate configuration is tested because of the wireless channel effects. The simulator provides us with such a possibility. The usage of a simulator, however, does not mean that different wireless channel effects (similar to those in a real network) cannot be modelled in the simulator (see wireless channel models in [8]).

2.2.2 Simulator calibration

In order to get realistic results, the operator needs to calibrate an energy consumption model and a wireless channel model in accordance to the environment where a network will be deployed (we calibrated the wireless channel for two specific environments in [4]). Their calibration can be done manually or automatically by the integration of a simulator and a testbed [5]. The calibration should take place before the optimization. The carefully calibrated wireless channel model (energy consumption model) can statistically reflect the wireless channel behaviour (energy consumption) in a real network.

2.2.3 Solution robustness

A wireless environment is dynamic. It can change in an unexpected way which may result into conditions that were not observed during the calibration. The framework, however, should cope with that, i.e., a found solution should keep working (preferably decreasing its effectiveness only gradually) even if some network characteristics change in the network. In order to achieve this, a candidate solution should be evaluated on networks with different topologies and a wireless channel model should be calibrated for different environments (e.g., network congestion, network maximum throughput). Moreover, the network operator can calibrate a wireless channel model for some pessimistic scenario (even though it has not been observed yet in a given environment). For example, he can deliberately increase the noise level in the noise model, increase the path loss or variation in the log-normal shadowing model. For more information on the log-normal shadowing model, see [8]. The evaluations obtained from different networks can be combined into a single evaluation score.

2.2.4 Framework generality

The framework is generic, and it could be used to optimize different types of an IDS (misuse-based or anomaly-based, centralized or distributed). Further, we list several examples.

In [14], the authors proposed a scheme that activates IDS agents preloaded in sensor nodes with a certain probability. Our framework can be used to find an optimal value of the probability for a given application scenario, i.e., to solve the trade-off between the number of packets (links) left unmonitored (influences a detection accuracy) and the number of IDS agents being activated (influences energy consumption).

In [16], the authors proposed a distributed IDS that involves a set of rules to detect different types of an attack. Our framework can be used to automatically select the appropriate rules and optimize (among others) their detection thresholds.

In [15], the authors proposed an IDS to detect packet reception rate and receive power anomalies. The authors demonstrated that there is a trade-off between detection probability, detection delays and false positives for their technique. Our framework can help a network operator to find the optimal values of the IDS parameters for his/her application scenario.

3 Implementation of the framework

Relevant high-level implementation details are provided in this section.

3.1 Optimization engine

There are two classes of optimization algorithms – exact and approximate (heuristic) algorithms. Since the evaluation of candidate solutions cannot be done analytically in our case, the exact algorithms can hardly be applied. The heuristics are divided into population-based and single-solution based algorithms. We use a population-based algorithm because in comparison to a single-solution based algorithm it provides us with the ability to evaluate multiple candidate solutions in parallel and hence to speed up the convergence of an optimization.

There is a variety of population-based algorithms, e.g., evolutionary algorithms (EAs), particle swarm optimization, immune networks. We use EAs as we already have successfully applied these algorithms for the automatic generation of secrecy amplification protocols in WSNs [3].

EAs work with a population of candidate solutions (*individuals* in terms of EAs) and *evaluate* them using a *fitness function*. EAs generate new candidate solutions applying genetic operators of crossover and mutation to the solutions in the population. In each *generation*, EAs update the population with new candidate solutions. The process repeats until an optimal solution is found or the maximum number of iterations is exceeded. For more information on EAs, see [7].

The optimization engine is based on Evolving Objects [9], an advanced component-based framework with a high number of already implemented optimization algorithms. For the purpose of this work, we used a basic evolutionary algorithm *eoEasyEA* that is highly configurable and suits our needs. We reused existing operators for selection, replacement, termination and statistics collection, and implemented only problem specific parts of initialization, mutation, crossover, and evaluation. More details on the settings of EAs can be found in Appendix.

3.2 Network simulator

We use the MiXiM network simulator [10], which is based on the OMNeT++ simulation framework [11]. MiXiM has a modular architecture with a high number of already implemented models for a WSN simulation. It inherits many advanced features from

OMNeT++ and thus is very adaptable and configurable. The whole simulation is configured via a dedicated OMNeT++ configuration file. A candidate solution (an individual) is represented as a list of configuration values stored in a separate configuration file. Before the evaluation (simulation) starts, the file is included in the main configuration file.

The choice of a general-purpose WSN simulator allowed us to move one step forward towards more realistic simulations, since MiXiM provides more accurate simulation models, e.g., for wireless channel, radio, and MAC layers, in comparison to a very fast purpose-built simulator we used in our previous work [3]. However, the accuracy comes at the price of speed. We simulated a network operating for one hour, and it took about 5 minutes to simulate such network on a single CPU core. In order to get a solution in acceptable time, we decided to utilize distributed computing.

We chose the BOINC distributed computing platform [12] for our experiments. In cooperation with the Institute of Computer Science at Masaryk University, we attached about 200 CPU cores from the campus to our BOINC infrastructure and used them when they were idle. Other 700 cores were available from the National Grid Infrastructure project MetaCentrum.

3.3 Configuration evaluation

A candidate configuration of an IDS is evaluated based on its *accuracy* and *memory usage*. In this technical report, two terms IDS and monitoring node are used interchangeably.

Notation 1. *The set* $A = \{a_1, ..., a_{n_m}\}$ *is a set of malicious nodes in a network.*

Notation 2. *The set* $C = \{c_1, ..., c_{n_b}\}$ *is a set of all benign nodes in a network.*

Notation 3. *The function* $x : N \rightarrow N$ *takes a sensor node index as an argument, and returns a number of the neighbours that consider this node benign.*

Notation 4. *The function* $y : N \rightarrow N$ *takes a sensor node index as an argument, and returns a number of the neighbours that consider this node malicious.*

Notation 5. *The function* $n : N \rightarrow N$ *takes a sensor node index as an argument, and returns a number of the neighbours of this node.*

Notation 6. *The function* $m : N \rightarrow N$ *takes a sensor node index as an argument, and returns the amount of memory (in bytes) used by an IDS on this node.*

3.3.1 Accuracy

We measured accuracy based on the number of true positives and true negatives:

- A true positive occurs when a monitoring node $c \in C$ correctly considers its neighbour $a \in A$ malicious.
- A true negative occurs when a monitoring node $c_i \in C$ correctly considers its neighbour $c_j \in C$ ($i \neq j$) benign.

A node $k \in C \cup A$ considers the node $l \in C \cup A$ as a neighbour if it received at least one packet from l during the simulation. We assume that sensor nodes are distributed in such a way, that every node in the network has at least one neighbour.

For a benign node c_i, we calculated the percentage of the neighbours that considered the node benign. Further, we found the average of such values over all benign nodes in the network and denoted the result as tn. Similarly, for a malicious node a_i, we calculated the percentage of the neighbours that considered the node malicious. Further, we found the average of such values over all malicious nodes in the network and denoted the result as tp.

The accuracy function is the weighted mean of tn and tp:
$\frac{w_1 * tn + w_2 * tp}{(w_1 + w_2)}$, where $tn = \frac{1}{|C|} * \sum_{c_i \in C} \frac{x(c_i)}{n(c_i)}$, $tp = \frac{1}{|A|} * \sum_{a_i \in A} \frac{y(a_i)}{n(a_i)}$.
We assume that $|C| > 0$ and $|A| > 0$.

The function values range from 0 to 1. If every malicious node in the network is detected by all of its neighbours, and every benign node in the network is not considered malicious by any of its neighbours, the accuracy function is equal to 1, i.e., the maximum possible value. On the other hand, if none of malicious nodes is detected by at least one of its neighbours, and every benign node is considered malicious by all of its neighbours, the accuracy function is equal to 0, i.e., the minimum possible value.

The proposed function does not take the distribution of $\frac{x(c_i)}{n(c_i)}$ and $\frac{y(b_i)}{n(b_i)}$ into account. In certain cases, e.g., when a base station uses a majority voting scheme to make a final decision whether a node is benign or not, it might be preferable to have more values of $\frac{x(c_i)}{n(c_i)}$ and $\frac{y(b_i)}{n(b_i)}$ that are slightly above 0.5 instead of a few values that are extremely high.

3.3.2 Memory usage

The effectiveness of memory usage by the IDS on a node $c_i \in C$ was evaluated using the formula: $\frac{1}{1+m(c_i)}$.

If the IDS is switched off at the node c_i, then $m(c_i) = 0$ and $\frac{1}{1+m(c_i)} = 1$. Furthermore, if $m(c_i)$ increases, then the effectiveness of memory usage decreases towards zero.

Further, we calculated the average value of $\frac{1}{1+m(c_i)}$ over all benign nodes in the network. More formally, it can be written as: $\frac{1}{|C|} * \sum_{c_i \in C} \frac{1}{1+m(c_i)}$. We assume that $|C| > 0$.

The designed function provides values that are not correlated to accuracy values. In certain cases, however, it might be useful to take into account that even a small amount of memory is a waste if the accuracy of an IDS is low. Yet a higher amount of used memory can be justified if the IDS is highly accurate.

3.3.3 Fitness function

For the purpose of this work, we added both accuracy and memory usage metrics together, making the accuracy metric to contribute more to the value of the sum than the memory metric by introducing weights. The weight was set to 1 for the accuracy metric, and it was set to 0.1 for the memory usage metric. The weight for the memory usage should be carefully selected – if it is too high (i.e., it is more important to save memory than to detect attacks), the optimal solution is to switch all IDSs off, or set a maximum number of monitored nodes and buffer size to zero. We set $w_1 = w_2 = 1$ in the accuracy metric. The resulting fitness function is:

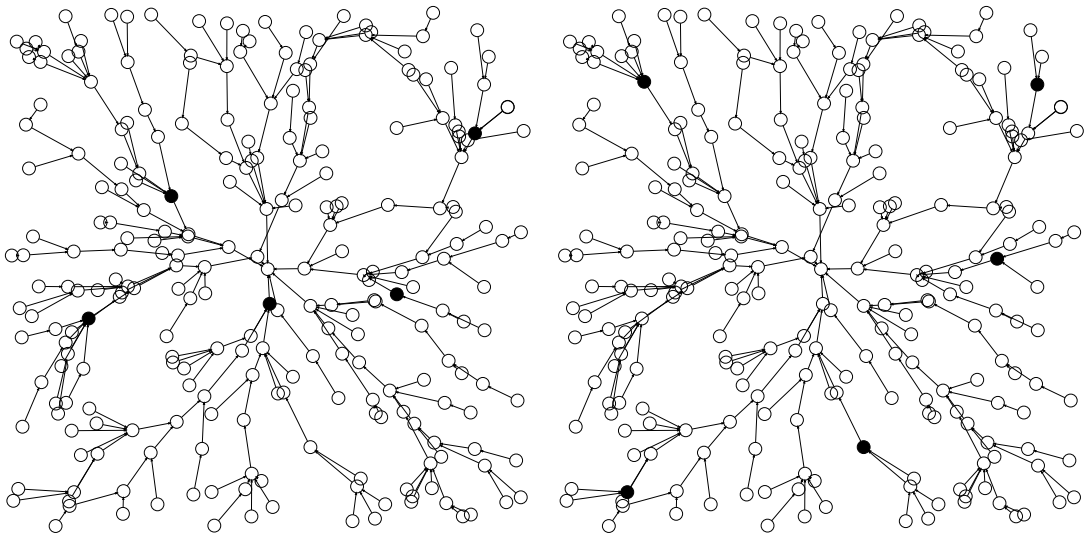
$\frac{1}{2|C|} * \sum_{c_i \in C} \frac{x(c_i)}{n(c_i)} + \frac{1}{2|A|} * \sum_{a_i \in A} \frac{y(a_i)}{n(a_i)} + 0.1 \frac{1}{|C|} * \sum_{c_i \in C} \frac{1}{1+m(c_i)}$.

4 Our test case

In this section, we describe the scenario for which we would like to test the framework on. The framework is used to find optimal parameters of an IDS (we implemented it in the MiXiM simulator for purposes of our previous work [6]) for a given scenario. A network operator (a person who uses our framework) knows behavior of benign nodes and assumes behavior of malicious nodes. If another than assumed type of an attack appears in the deployed network, the parameters found by the framework might not be optimal for such network.

4.1 Topology

We generated a topology of 250 static sensor nodes uniformly randomly distributed over an 200 m $\times$ 200 m area. A single base station is placed in the center of the area.



(a) 1st placement of mal. nodes. (b) 2nd placement of mal. nodes.

Figure 2: Topology, routing tree, and placement of malicious nodes.

The topology together with a routing tree is depicted in Figure 2. There are 246 benign sensor nodes (white) including the base station, and 5 malicious nodes (black filled). According to [13], the terrestrial WSNs typically consist of hundreds to thousands of inexpensive sensor nodes. However, the purchase of a large network is not always feasible due to the current price of sensor nodes. A MICAz sensor node costs about €80. Therefore, we believe that medium-sized networks that consist of hundreds of nodes are more reasonable to consider.

In order to make the analysis of results (w.r.t. their optimality) from the framework simpler and more intuitive, we focus on static sensor nodes. However, more dynamic network scenarios can be modeled as well. MiXiM provides a variety of node mobility models [10].

4.2 Benign node

We assume a benign node uses a network stack that consists of application, routing protocol, MAC protocol, and intrusion detection system.

4.2.1 Application layer

We consider a standard application for a WSN, where every node sends one packet to a base station every 30 seconds. The application runs for one hour. In order to avoid collisions (due to node synchronization), the whole time-frame is divided into intervals

of length 30 seconds. For every interval i, a node generates a random number r ($0 \leq r \leq 30$) and starts transmitting at $i + r$. The size of a packet is 152 B.

4.2.2 Network layer

We assume that the network layer uses static routing. The routing tree was generated as follows. A base station broadcasts a packet containing its identification together with the value h set to 0. A node waits until it receives a packet from a neighbour that is the closest one (has the highest signal strength). Then the node sets the neighbour as its parent, increases value h by 1 and broadcasts the value together with its identification. Value h represents number of hops to the base station.

4.2.3 Medium access layer

We use CSMA-CA at this layer.

4.2.4 Physical layer

We use a model for CC2420 radio that is commonly used in sensor nodes, e.g., TELOSB, MICAz sensor nodes.

4.2.5 Intrusion detection system

We use a simple IDS that we implemented in the MiXiM simulator for the purpose of our previous work [6]. The IDS uses a detection rule (more specifically, a retransmission rule) from [16]. It is not the goal of this work to propose a complex IDS, but rather to test the framework as such. In the conventional networks, one can use a commercial IDS, to run the framework on, and compare the IDS before and after the optimization to see the improvement. For WSNs, however, to our best knowledge, there are no commercial IDSs available.

An IDS is running on a sensor node and it continuously analyzes sent and overheard packets. The IDS does not include responsive and collaborative components (see [20] for the conceptual architecture of an IDS in WSNs). Therefore, the IDS does not generate any additional traffic.

A monitoring node overhears to some extent both incoming and outgoing packets of a close enough monitored neighbour. An IDS stores a table, where each row corresponds to a certain monitored node. The table contains the number of packets received

(PR) and forwarded (PF) by a monitored node. The number of rows is limited to a *number of monitored nodes*.

The detection exploits the fact that a monitoring node overhears (to some extent) both incoming and outgoing packets of a close enough monitored neighbour. If the IDS on a node $c_i \in C$ overhears a packet P sent to a node $b_j \in C \cup A$, b_j is close enough and b_j should forward the packet (e.g., b_j is not a base station), then the IDS stores P in the buffer and increments the PR counter of the monitored node b_j. The number of packets is limited by a *buffer size*. If a new packet arrives but the buffer is full, the oldest packet is removed from the buffer. When the IDS overhears the packet P being forwarded by the node b_j, it removes P from the buffer (if it is still there) and increments the PF counter of the node b_j. Since both the table and the buffer are limited, the IDS monitors only the closest nodes and the newest packets.

The detection is done at the end of the simulation based on the collected statistics. The node c_i considers the node b_j as a selective forwarder if the dropping ratio of b_j, i.e., ratio of a number of packets dropped to a number of packets received, is higher than a predefined *detection threshold*. If the node c_i overheard less than the predefined *number of packets received* by b_j as overheard by c_i, c_i does not consider b_j malicious as the number of overheard packets is small and there is a high level of uncertainty. We cannot influence the number of overheard packet, but we can change our decision making process based on this value and potentially decrease a number of false positives.

The IDS running on a node c_i consumes $m(c_i) = p_1 * 8 + p_2 * 16$ B of memory. Each record in the table occupies 8 B (4 B for node ID, 2 B for PR, 2 B for PF), and there are p_1 such records. Each record in the buffer occupies 16 B (4 B for MAC source ID, 4 B for MAC destination ID, 4 B for MAC intermediate node ID, and 4 B for packet counter), and there are p_2 such records.

We would like to optimize the following four parameters: p_1 (number of nodes to be monitored), p_2 (a number of packets stored in a buffer), p_3 (number of packets received) and p_4 (detection threshold). The value of p_1 ranges from 0 to 54 (the maximum number of neighbours in our simulation scenario), p_2 – from 0 to 100, p_3 – from 0 to 2000, and p_4 from 0 to 100.

4.3 Malicious node

Currently, for our proof-of-concept implementation of the framework, we assume a single type of malicious node – a selective forwarder, i.e., a node that drops a certain per-

centage of received packets. We assume the model is the same as the model of a benign node, except for the network layer that is modified to drop a certain percentage of received packets (in our case 50%), and an IDS that is omitted.

5 Testing results

We tested our prototype on three optimization scenarios, each with the different size of the search space. Their description together with the obtained results are presented in the following subsections. The settings of EAs for each optimization scenario are described in the corresponding subsections of Appendix. For the evaluation of a candidate configuration, we used the fitness function defined in Subsection 3.3. Time needed to complete the optimization is indicated in terms of a number of EA generations and evaluations.

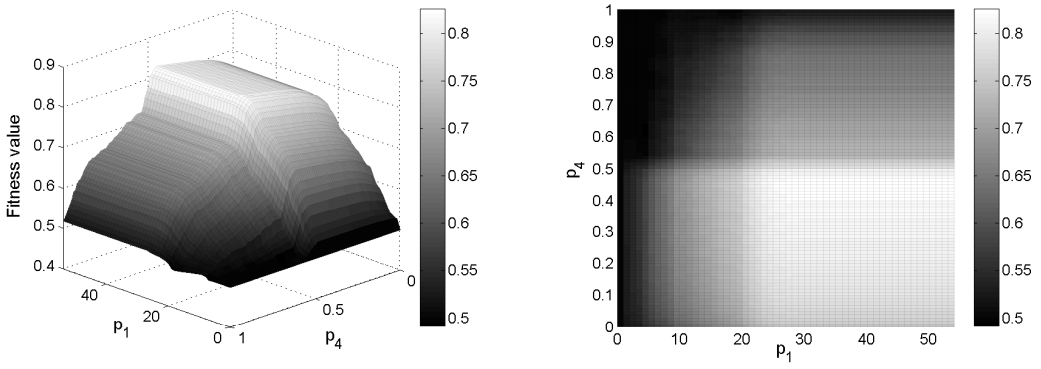
5.1 Optimization scenario No. 1

In this scenario, we assume that every benign node in the network runs an IDS, and the IDS is configured in the same way for these nodes. The goal is to optimize the configuration (p_1, p_2, p_3 and p_4), common for all sensor nodes, using our framework.

We performed both an exhaustive search and an EA-based search, and compared the results. In order to make the exhaustive search timely acceptable, we fixed $p_2 = 100$ and $p_3 = 0$. The reduced search space contained 5555 possible configurations (see the description of an IDS in Subsection 4.2).

5.1.1 Exhaustive search

Fitness values for each possible combination of p_1 and p_4 are depicted in Figure 3. The maximum fitness value (0.8249276442) was achieved for the configuration with $p_1 = 27$ and $p_4 = 0.45$. The threshold is below 0.5 (the dropping rate of a malicious node, see Subsection 4.3), because a monitoring node cannot reliably overhear all packets sent to a malicious node, and hence the dropping rate of the malicious node as observed by the monitoring node may be lower than 0.5.



(a) View from the side.

(b) View from the top.

Figure 3: Fitness values for all possible combinations of p_1 and p_4.

5.1.2 Evolutionary algorithm

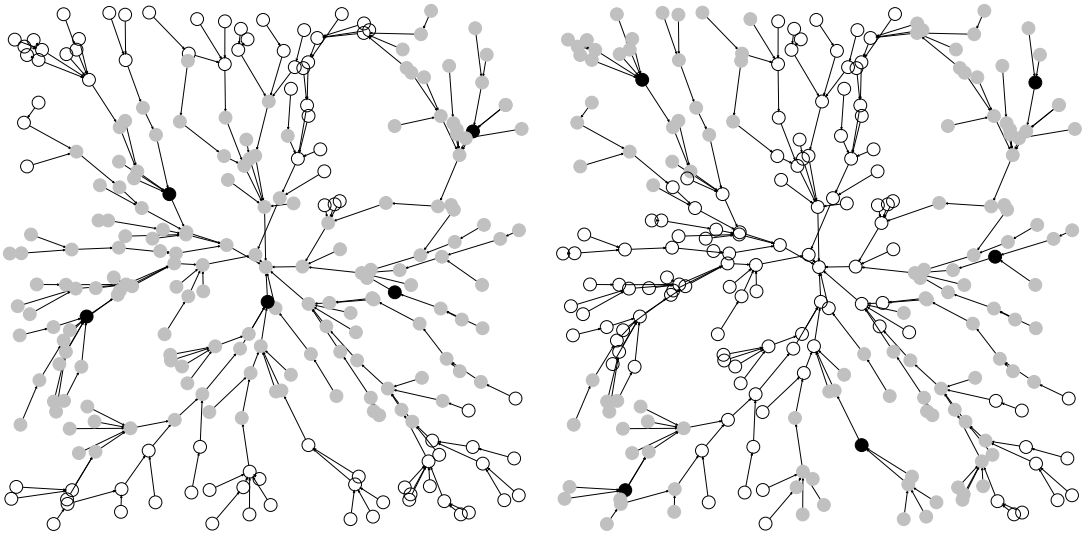
We ran the optimization process 30 times. The EA was able to find the best configuration (found by the exhaustive search) using 19 generations on average. The standard deviation was 9.64. On average, the EA required 144.87 evaluations. The standard deviation was 67.37. In the worst case, the EA required 271 evaluations, while the exhaustive search required 5400 evaluations.

5.2 Optimization scenario No. 2

In this scenario, we assume that only a subset of benign nodes runs IDSs, which are configured in the same way on all selected nodes. As opposed to the node (IDS) placement problem, the framework should find the optimal placement as well as the optimal configuration of the IDSs.

There are 250 parameters to optimize: p_1, p_2, p_3, p_4, and 246 Boolean parameters, each indicating whether the IDS should be enabled or disabled on a given node. We fixed two parameters ($p_2 = 100$ and $p_3 = 0$). The search space contained $55 * 101 * 2^{246}$ possible configurations.

We ran the optimization process on two networks with the same topology but with the different placement of malicious nodes (see Figure 4(a) and Figure 4(b)). The malicious nodes are depicted with black filling. We repeated the optimization process 30 times for both placements of malicious nodes.



(a) 1st placement of mal. nodes. (b) 2nd placement of mal. nodes.

Figure 4: The best found placements of IDSs in the network.

5.2.1 First placement of malicious nodes

The best configuration found by the EA had the fitness value equal to 0.8940953359, $p_1 = 27$, and $p_4 = 0.45$ (the same values were found for the first optimization scenario). The switched on/off IDSs are depicted in Figure 4(a). Nodes that ran an IDS are depicted with grey filling. The configuration generated 1723 false positives and 33 false negatives, i.e., 7.03 false positives per benign node, and 6.6 false negatives per malicious node. The configuration was found using 581.77 generations on average. The standard deviation was 88.59. On average, the EA required 10600.17 evaluations. The standard deviation was 1603.29. In the worst case, the EA required 13349 evaluations.

Although we did not perform an exhaustive search, we believe that the found configuration was optimal. The intuition behind this is as follows. If an IDS running on a node $c_i \in C$ detects a malicious neighbour $a_j \in A$, then switching it off reduces a number of false positives, increases memory usage effectiveness, but causes a false negative. As we discovered, it is natural for the EA to switch such an IDS on since the benefit (according to the designed evaluation function, see Subsection 3.3) is higher than from switching the IDS off. We verified (by setting $p_4 = 0$ and $p_1 = 54$) that the EA achieved the minimum possible number of false negatives, and the minimum number of false positives for the given number of false negatives.

5.2.2 Second placement of malicious nodes

The best configuration found by the EA had the fitness value equal to 0.8780288967, $p_1 = 24$, and $p_4 = 0.5$. The switched on/off IDSs are depicted in Figure 4(b). The configuration generated 1173 false positives and 42 false negatives. We believe that the found configuration was optimal. The intuition behind this is the same as for the first placement of malicious nodes. We verified (by setting $p_4 = 0$ and $p_1 = 54$) that the achieved number of false negatives was the lowest possible, and the achieved number of false positives was the lowest possible for the given number of false negatives.

The best configuration was found by 14 optimization runs. The average number of generations was 499.57. The standard deviation was 82.185. On average, the EA required 9096.1 evaluations. The standard deviation was 1501.5. Other 8 runs ended up with the configuration where the fitness value was equal to 0.8764108606. The rest of the runs found configurations with fitness values equal to or higher than 0.8656972029.

When comparing the best configurations found for both placements, an average number of false events per node (an average number of false positives per node added together with an average number of false negatives per node) was higher for the first placement (13.6 versus 13.17), but the fitness value was higher for the second one (0.8940953359 versus 0.8780288967). That might be caused by the fact that nodes falsely accused or falsely not detected in the first placement had a higher number of neighbours, which is also reflected in the selected value of p_1 (27 versus 24).

5.3 Optimization scenario No. 3

In this scenario, we again assume that only a subset of benign nodes runs IDSs. In comparison to the previous optimization scenario, here each IDS may be configured in a different way. As opposed to the node (IDS) placement problem, the framework should find the optimal placement as well as the optimal configuration of each IDS.

The search space contained $(55 * 101 * 2001 * 101 * 2)^{246}$ configurations. The IDS implemented in our test case did not influence the network traffic, and hence did not influence other IDSs. Therefore, configurations of any two IDSs were independent of each other. We launched 246 independent optimizations and significantly reduced the search space, i.e., to $246 * (55 * 101 * 2001 * 101 * 2)$ possible configurations.

We did two experiments. In the first experiment, we launched a single optimization that searched for the best configuration for all sensor nodes together. In the second

experiment, we launched 246 independent optimizations, each searching for the best configuration for a particular sensor node only.

5.3.1 First experiment

We started with a randomly generated initial population. The evolution began to improve the configuration, but the speed of convergence was too slow. Therefore, we decided to start the optimization process again, using the population of the best configurations from the second optimization scenario (see Subsection 5.2). This optimization was gradually improving the configuration and reached the fitness value of 0.955133 after 136′765 evaluations in the $21'439^{\text{th}}$ generation. The evolution was stopped when no improvement was found during next 500 generations.

5.3.2 Second experiment

All 246 optimizations started with the initial population that contained the best configurations for the second optimization scenario. By combining optimized parameters from these optimizations, a configuration that reached the fitness value of 0.9604364302 was found. When compared to the second optimization scenario, this approach was able to significantly improve the resulting IDS configuration by tweaking the parameters independently for each node. A configuration found in each independent optimization required 422.63 generations on average. The standard deviation was 63.84. On average, the EA required 2811.68 evaluations to find a configuration in each independent optimization. The standard deviation was 427.05. In the worst case, the EA required 5365 evaluations.

We ran another experiment to verify whether the configuration found by the EA was optimal. The experiment was based on the intuition we mentioned in Subsection 5.2. For an IDS, we fixed the parameters in such a way that it could detect as many malicious nodes as possible ($p_1 = 54$, $p_2 = 100$, $p_3 = 0$, $p_4 = 0$). Further, we found the minimum value of p_1 such that all malicious nodes can still be detected by the IDS. The same procedure was repeated for other three parameters. We confirmed that p_1, p_2 and the placement of IDSs were optimal. However, p_3 and p_4, as we discovered, could be further improved.

6 Related work

The aim of our work is to provide a generic framework for optimization of intrusion detection techniques. Despite evaluating our framework using a particular detection technique for selective forwarding attacks, we did not propose a particular novel technique for intrusion detection in WSNs. Some of the detection techniques that can be optimized using our optimization framework can be found, e.g., in [16, 15, 14, 17].

There are various optimization techniques that can assist in WSN design and usage. Several papers have been published regarding the placement of nodes (IDSs) that can be considered as a subproblem of a more general node (IDS) configuration problem where one of the node (IDS) configuration parameters may indicate where the node is placed (whether an IDS is enabled/disabled at this node).

The node placement problem received a considerable attention from the research community since it has significant impact on the functionality of WSNs and influences the detection accuracy of IDSs as well. For example, in a sparse topology, a node can monitor only a limited number of packets in its neighbourhood. 46 techniques that help to find optimal node placement in WSNs are surveyed in [19]. In contrast, our framework is more generic and additionally provides the possibility to optimize a configuration of IDS parameters (e.g., detection threshold, buffer size). For more information, see Sections 4 and 5. To the best of our knowledge, we are the first designing a tool for optimization of IDS parameters in WSNs.

The placement techniques can be categorized into two main groups: *static* and *dynamic* [19]. The primary objectives are specified as follows: area coverage, network connectivity, network longevity and data fidelity where the network connectivity and longevity is particularly important for the detection accuracy and lifetime of an IDS.

The *static techniques* deal with situations where nodes are placed prior to the network start-up either deterministically or randomly. While the deterministic deployment implies a full control of the topology, the random deployment allows one to control it only to some extent. Random deployment techniques are further categorized in [26]. The more controlled deployment is performed, the easier IDS placement/activation is. We consider static uniformly distributed topology in our work.

The *dynamic techniques* are classified in [19]. *Post-deployment repositioning* means that, e.g., coverage improvement is needed after random node distribution, and *on-demand repositioning* means that, e.g., improvement of certain performance dependent

on changes in the environment is needed. All such dynamic changes have to be taken into account by a network operator when considering IDSs deployment. The MiXiM simulator enables a dynamic reorganization of the WSN. Thus, these techniques can straightforwardly extend our current framework.

Relay sensor node placement problem is considered in [27]. The goal is to compute a minimum number of relay nodes (such that they form a globally connected topology) for a given set of duty sensor nodes in the area and upper-bounded transmission range of a sensor node. The work is based on [35] where Steiner trees are minimized with a minimum number of Steiner points. The relay sensor nodes act as Steiner points. Configuration of relay sensor nodes is not considered.

Issues of cooperative IDSs in resource constrained wireless (ad hoc or sensor) networks are investigated in [24]. The authors aimed at detection of the following attacks: 1) flooding, 2) port scanning, and 3) web exploits. In our framework, we consider more relevant attacks aimed at WSNs. The authors of [24] considered trade-offs among different objectives that are calculated using the following models. The *power model* is used to measure energy consumption that is based on the current role of each of the nodes. The *information model* is used to measure entropy of the nodes' observations. The *delay model* is used to measure delays caused by aggregations of the reports. Finally, the *network coverage model* is used to measure coverage of the sensor nodes by the IDS nodes. Similarly to our approach, all the objectives were blended into a single fitness function and optimized using an EA.

Anjum et al. in [22] proposed an algorithm finding a minimum number of activated IDSs such that every packet forwarded from a source towards the base station was analyzed at least once on its path. An IDS could be activated only on tamper-resistant sensor nodes that formed only a part of a heterogeneous WSN. The authors suggested to divide the WSN into clusters with a tamper-resistant sensor node as a clusterhead. The goal was to find out a *minimum cut-set* that was a minimum set of tamper-resistant sensor nodes such that packets from all sources traversed at least one node of this set. In comparison to our work, the authors did not consider IDS configuration. Furthermore, the algorithm considers only packets forwarded by the monitoring nodes and not packets overheard in a promiscuous mode.

The authors of [28] introduced utilization of EAs for the node placement problem in WSNs, where the placement of sensors in chemical plants to effectively monitor linear mass flow processes was optimized. The optimization aimed at covering all streams

with respect to cost, accuracy and reliability. In comparison to our work, the communication between the sensor nodes was not considered. The algorithm combined concepts of graph theory and EAs.

The authors of [29] aimed at multi-objective optimization using an EA for deployment of a homogeneous WSN with the coverage and lifetime of the network as objectives. They further discussed changes in the layout for different ratios of sensing to communication ranges.

Ferentinos and Tsiligirids in [25] presented a methodology of multi-objective optimization for self-organizing WSNs using an EA. Their fitness function took application-specific, connectivity and energy-related metrics into account. Similarly to our work, multiple objectives of the optimization problem were blended into a single objective function using weighted sum. An optimized WSN consisted of nodes that could operate in the following modes: 1) *active mode with a high transmission range*; 2) *active mode with a low transmission range*; 3) *inactive mode* and 4) *clusterhead mode*. The goal was to find out optimized operation mode for each node in the network. The authors did not consider any IDS. Furthermore, they did not consider node parametrization.

Methodology how a multi-objective evolutionary algorithm for design-space exploration could be configured was presented in [30]. The authors divided the design-space into the network-level configuration space and the node-level configuration space where the former was defined for (global) network-level parameters and the latter for (local) node-level parameters. An individual for the EA was a vector of $N + 1$ integer values where N elements referred to the configuration of N nodes in the WSN and 1 element referred to the network-level configuration. A case study of the authors works with two node-level parameters (transmission power and sampling time) and two network-level parameters (frame length and number of active slots per frame). Lifetime, latency and reliability are used as three QoS (Quality of Service) metrics with several trade-offs. In our work, we used a different set of metrics. The authors used multi-objective optimization. We used single-objective optimization, and currently work on incorporating multi-objective optimization.

Khanna et al. used EAs for several optimization issues in WSNs ([31, 32, 33]). In [31], the authors aimed at minimizing power consumption while maximizing the coverage and exposure by switching the sensor nodes to following states: 1) *inactive sensor node*; 2) *active sensor node*; 3) *cluster-head* and 4) *inter-cluster router*. Consequently, the routes toward the base station are optimized according to the topology. The gene contains the

identification of each sensor and corresponding configuration. In the *initialization phase*, temporary clusters of the sensors with domain specification are created. In the *adaptation phase*, the network is adapted according to the fitness function to create accurate cluster boundaries and to create optimal routes to the base station. Since the authors identified two competing objectives (cluster membership that changes dynamically because of dead or depleted sensor nodes and routes that change because of high-cost paths avoidance), they use a multi-objective EA. The fitness functions are following: 1) *Total node fitness* (TNF) and 2) *Total route fitness* (TRF). In TNF, the following measures are used: 1) uniform distribution of clusters (number of nodes in each of the cluster); 2) power needed to enable communication based on path loss computation; 3) battery status based fitness and 4) inter-cluster router load. In TRF, where the goal is to generate balanced routes based on the node fitness function, the following measures are used: 1) power needed to communicate; 2) battery status, and 3) bit-rates of mutual communications. No wireless channel model is used for the simulations of communication between nodes. In contrast, our framework assumes a careful calibration of the wireless channel model to provide realistic configurations and uses completely different metrics.

In [32], Khanna et al. presented an approach on a WSN deployment showing that it can be optimized dynamically to allow for sensor nodes mobility and considered security aspects as well. Their previous work [31] was extended by adding mobility and security fitness functions. *Sensor position* is dynamically adapted based on optimal coverage prediction. *Sensor security* dynamically enables encryption and authentication based on security threats or network traffic. The *mobility extension* covers following aspects: 1) coverage uniformity fitness, 2) cluster-node migration fitness, 3) cluster-head migration fitness, 4) node motion fitness and 5) sensor data fitness. The *security extension* utilizes "secure node fitness" that is used to decide whether the security features should be enabled or not. The security threats are evaluated by the base station based on a number of bad packets and energy requirements. In the simulation, each node is given simply a coverage area of 3×3 units of distance and the traffic is simulated in the ns-2 simulator. However, the information on wireless channel model and other details are not specified. In our work, we use a realistic wireless channel model where the packet delivery ratio depends on proximity of the sensor nodes, noise level, interferences, etc.

Finally, in [33], Khanna et al. incorporated *local monitoring nodes* (LMNs) into the WSN. These nodes observe suspicious behavior and monitor data message patterns message collisions, route traffic activity trends and sensor positioning in their neigh-

bourhood. The fitness function measures optimality of the LMN positioning and accurate identification of malicious nodes. This fitness function competes with fitness functions presented in [32]. The target node is evaluated by the LMN for the following aspects: 1) the received signal strength, 2) transmission periodicity, 3) spurious transmission from nodes that are not listed as neighbours, 4) response delay (or unresponsiveness) and 5) packet dropping or modification. In the experiment, the authors focused on the malicious node detection rate and the false positive node detection rate based on a variable number of malicious node deployment, where the ratio of malicious nodes in the WSN varied between 0.05 to 0.30. The work of Khanna et al. [33] is the only work on optimization of IDSs using EAs for WSNs, except for our IDS framework. However, in their work, the authors do not consider optimization of IDSs parameters.

7 Conclusion and future work

To our best knowledge, there is no work that focuses on (semi) automatic and systematic configuration of intrusion detection systems for wireless sensor networks, which we believe is an important area to explore.

In this work, we describe procedures to optimize the configuration of an intrusion detection system for a given application of a wireless sensor network. Also, we discuss how solution robustness and solution realism can be achieved.

We presented a prototype of our framework that optimizes the configuration of an intrusion detection system in wireless sensor networks. The design and implementation of the framework leveraged our previous results in the field of simulators (particularly realism of simulators for wireless sensor networks), optimization and evaluation metrics.

We tested our framework on three carefully selected scenarios with a different size of their search space. Our results demonstrated that evolutionary algorithms can be potentially used to search for a solution of the given problem more effectively. However, this conclusion is not valid in general (since the hypothesis was tested only on three selected scenarios). More general conclusion can be made if the community use evolutionary algorithms on a bigger set of different scenarios (different application, routing, medium access control and physical layer, different topologies, different attacker models).

Our framework can find reasonable (if not optimal) configuration of an intrusion detection system for a given (arbitrary but specified in advance) scenario. Values found

by our framework may not be reasonable if these values are used in a different scenario, i.e., intrusion detection system, topology, attacker behavior.

The obtained results, we believe, are more than promising. Hence, we plan to use our framework to optimize different techniques for detection of different attackers. Also, we plan to use our framework to optimize an intrusion detection system for our laboratory testbed, configure the intrusion detection system according to the output provided by the framework, deploy it, and analyze the results collected from the testbed.

The future version of our framework will also use multi-objective evolutionary algorithms.

While we focused on the optimization of an intrusion detection system (other layers were fixed), we believe that such a framework can be easily extended to optimize the whole network stack. This can be explored in the future.

Acknowledgment

This work has been undertaken primarily with equipment from the Czech Security Research project VG20102014031. We thank the National Grid Infrastructure project MetaCentrum and Institute of Computer Science at Masaryk University for the provided computational resources.

References

[1] Texas Instruments. CC2420 – 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF transceiver. [Online]. Available at: http://focus.ti.com/.

[2] Stetsko A., Matyas V.: Effectiveness metrics for intrusion detection in wireless sensor networks. European conference on computer network defense. (2009) 21–28.

[3] Svenda P., Sekanina L., Matyas V.: Evolutionary design of secrecy amplification protocols for wireless sensor networks. ACM conference on wireless network security. (2009) 225–236.

[4] Stetsko A., Stehlik M., Matyas V.: Calibrating and comparing simulators for wireless sensor networks. IEEE conference on mobile adhoc and sensor systems. (2011) 733–738.

[5] Wen Y., Zhang W., Wolski R., Chohan N.: Simulation-based augmented reality for sensor network development. ACM conference on embedded networked sensor systems. (2007) 275–288.

[6] Stetsko A., Smolka T., Jurnecka F., Matyas V.: On the credibility of wireless sensor network simulations: evaluation of intrusion detection system. Conference on simulation tools and techniques. (2012) 75–84.

[7] Talbi E.-G.: Metaheuristics – From Design to Implementation. John Wiley & Sons, Inc. (2009).

[8] Rappaport T.: Wireless communications: Principles and practice, 2nd ed. Prentice Hall PTR. (2001).

[9] Keijzer M., Merelo J. J., Romero G., Schoenauer M.: Evolving objects: A general purpose evolutionary computation library. Conference on evolution artificielle. (2002) 231–242.

[10] Kopke A., Swigulski M., Wessel K., Willkomm D., Haneveld P. T. K., Parker T. E. V., Visser O. W., Lichte H. S., Valentin S.: Simulating wireless and mobile networks in OMNeT++ the MiXiM vision. Conference on simulation tools and techniques for communications, networks and systems & workshops. (2008).

[11] OMNeT++ Community: http://www.omnetpp.org/.

[12] Anderson, D.P.: BOINC: a system for public-resource computing and storage. IEEE/ACM workshop on grid computing. (2004) 4–10.

[13] Yick J., Mukherjee B., Ghosal D.: Wireless sensor network survey. Computer networks, vol. 52(12), 2292-2330, 2008.

[14] Roman R., Zhou J., Lopez J.: Applying intrusion detection systems to wireless sensor networks. IEEE Consumer Communications and Networking Conference. (2006) 640–644.

[15] Onat I., Miri A.: An intrusion detection system for wireless sensor networks. IEEE conference on wireless and mobile computing, networking and communications. (2005) 253–259.

[16] da Silva, A. P. R. and Martins M. H. T. and Rocha B. P. S. and Loureiro A. A. F. and Ruiz L. B. and Wong H. C.: Decentralized intrusion detection in wireless sensor networks. ACM international workshop on quality of service & security in wireless and mobile networks. (2005) 16–23.

[17] Stetsko A., Folkman L., Matyas V.: Neighbor-Based Intrusion Detection for Wireless Sensor Networks. Conference on wireless and mobile communications. (2010) 420–425.

[18] Karlof C. and Wagner D.: Secure routing in wireless sensor networks: attacks and countermeasures. Ad hoc networks, 1(23). (2003) 293–315.

[19] Younis M., Akkaya K.: Strategies and techniques for node placement in wireless sensor networks: A survey. Ad hoc networks, 6(4). (2008) 621–655.

[20] Zhang Y., Lee W.: Intrusion detection in wireless ad-hoc networks. Conference on mobile computing and networking. (2000) 275–283.

[21] Roosta T., Pai S., Chen P., Sastry S., Wicker S.: Inherent security of routing protocols in ad-hoc and sensor networks. Global telecommunications conference. (2007) 1273–1278.

[22] Anjum F., Subhadrabandhu D., Sarkar S., Shetty R.: On optimal placement of intrusion detection modules in sensor networks. Conference on broadband networks. (2004) 690–699.

[23] Liu C., Cao G.: Distributed monitoring and aggregation in wireless sensor networks. Conference on computer communications. (2010) 1–9.

[24] Hassanzadeh A., Stoleru R.: Towards optimal monitoring in cooperative IDS for resource constrained wireless networks. Conference on computer communications and networks. (2011) 1–8.

[25] Ferentinos K. P., Tsiligiridis T. A.: Adaptive design optimization of wireless sensor networks using genetic algorithms. Computer networks, 51(4). (2007) 1031–1051.

[26] Ishizuka M., Aida M.: Performance study of node placement in sensor networks. In: Proceedings of the 24th International Conference on Distributed Computing Systems. (2004) 598–603.

[27] Cheng X., Du D. Z., Wang L., Xu, B.: Relay sensor placement in wireless sensor networks. Wireless Networks, 14(3). (2007) 347–355.

[28] Sen S., Narasimhan S., Deb, K.: Sensor network design of linear processes using genetic algorithms. Computers & chemical engineering, 22(3). (1998) 385–390.

[29] Jourdan D. B., de Weck O. L.: Layout optimization for a wireless sensor network using a multi-objective genetic algorithm. IEEE vehicular technology conference. (2004) 2466–2470.

[30] Nabi M., Blagojevic M., Basten, T., Geilen, M., Hendriks, T.: Configuring multi-objective evolutionary algorithms for design-space exploration of wireless sensor networks. ACM workshop on performance monitoring and measurement of heterogeneous wireless and wired networks. (2009) 111–119.

[31] Khanna R., Liu H., Chen H. H.: Self-organization of sensor networks using genetic algorithms. IEEE conference on communications. (2006) 3377–3382.

[32] Khanna R., Liu H., Chen, H. H.: Dynamic optimization of secure mobile sensor networks: A genetic algorithm. IEEE conference on communications. (2007) 3413–3418.

[33] Khanna R., Liu H., Chen, H. H.: Reduced complexity intrusion detection in sensor networks using genetic algorithm. IEEE conference on communications. (2009) 1–5.

[34] Heady R., Lugar G., Servilla M., Maccabe A.: The Architecture of a Network Level Intrusion Detection System. Technical report: University of New Mexico. (1990)

[35] Lin G. H. and Xue G.: Steiner tree problem with minimum number of Steiner points and bounded edge-length. Information Processing Letters, 69(2). (1999) 53–57.

A Settings of evolutionary algorithms

We start with the settings common for all three scenarios (see Section 5) and then proceed with the settings special for each of the scenarios in the corresponding subsections.

With each generation, the EA updated a population of individuals (configurations). The initial population was generated randomly if we did not state otherwise. The size of the population was S, and it remained constant for each generation. For each generation, a fraction $1 - R$ of the individuals with the highest fitness values was kept in

the population without modifications. The rest – R of the population – was replaced with individuals that were generated as follows. A stochastic tournament routine was executed $R * S$ times, each time selecting randomly two individuals from the original population, comparing their fitness values, and picking the better individual with the probability 0.9 or the worse one with the probability 0.1. The stochastic tournament increased the diversity of a population – a worse individual that might benefit the population in the future generation could be also included. The selected individuals (the fraction R of the population) were further transformed using crossover and mutation operations. With the probability *pCross*, we applied the crossover operation to each successive and not overlapping pair of individuals. If an individual contained n parameters, we decided to use $n - 1$ crossover points. We applied the mutation operator to each parameter of an individual with a certain probability *pMut*. The optimization iterated until the predefined condition held, e.g., parameters became optimal for a given scenario, or the maximum number of iterations was exceeded.

A.1 Optimization scenario No. 1

We ran the EA with the following settings. The size of the population S was set to 12. The fraction of replaced individuals during each iteration R was set to $\frac{2}{3}$. The probability of mutation *pMut* was the same for both parameters p_1 and p_4, and it was equal to 0.5. If a parameter mutated, we added or subtracted (based on the coin toss) an integer that was generated randomly from the discrete uniform distribution of values within the range from 0 to 5. The crossover operation was done with the probability *pCross* equal to 0.5. Since there were two parameters to optimize, we had a single point of crossover.

A.2 Optimization scenario No. 2

We ran the EA with the following settings. The size of the population S was set to 30. The fraction of replaced individuals during each iteration R was set to $\frac{4}{5}$. The probability of mutation *pMut* was equal to 0.1 for both parameters p_1 and p_4, and 0.004 for the rest – parameters that determined whether an IDS was switched on or off on a corresponding node. If p_1 or p_4 mutated, we added or subtracted (based on the coin toss) an integer that was generated randomly from the discrete uniform distribution of values within the range from 0 to 5. If other parameters mutated, we flipped their value, i.e., from 0 to 1, and vice versa. The crossover operation was done with the probability *pCross*

equal to 0.2. Because there were 248 (p_1, p_4, and 246 Boolean parameters) parameters to optimize, we had 247 crossover points.

A.3 Optimization scenario No. 3

We ran the EA with the different settings for each of the experiments (see Subsection 5.3):

A.3.1 First experiment

The size of the population S was set to 12. The fraction of replaced individuals during each iteration R was set to $\frac{4}{5}$. The probability of mutation *pMut* was the same for all $5 * 246 = 1230$ parameters, and it was equal to 0.000813. The crossover operation was done with the probability *pCross* equal to 0.5 and there were 1229 possible crossing points.

A.3.2 Second experiment

The size of the population S was set to 12. The fraction of replaced individuals during each iteration R was set to $\frac{4}{5}$. The probability of mutation *pMut* was the same for all 5 parameters, and it was equal to 0.2. The crossover operation was done with the probability *pCross* equal to 0.5 and there were 4 possible crossing points.