The Placement-Configuration Problem for Intrusion Detection Nodes in Wireless Sensor Networks

Juan E. Tapiador\textsuperscript{a,1,*}, John A. Clark\textsuperscript{b}

\textsuperscript{a}Department of Computer Science, Universidad Carlos III de Madrid, Avda. Universidad 30, 28911 Leganés, Madrid, Spain
\textsuperscript{b}Department of Computer Science, University of York, YO10 5GH, York, UK

Abstract

The deployment and configuration of a distributed network intrusion detection system (NIDS) in a large Wireless Sensor Network (WSN) is an enormous challenge. A reduced number of devices equipped with detection capabilities have to be placed on strategic network locations and then appropriately configured in order to maximise the detection rate and minimise the amount of computational and physical resources consumed – fundamentally, energy, which in turn depends on CPU, memory, and network usage. In practice, a major difficulty lies in the fact that the relationship between each node’s tuning parameters and the overall cost/benefit rate achieved by the deployment is poorly understood. We call this the Placement-Configuration Problem (PCP). In this paper we formalise and study this problem both theoretically and empirically. We introduce a formal model of distributed NIDS upon which the cost/benefit tradeoffs can be appropriately derived. Subsequently we show that, in general, the PCP is hard (NP-complete) and present a heuristic local search algorithm to find near-optimal solutions for practical scenarios. Our analysis framework is general in the sense that it is applicable to a number of existing detection technologies for WSNs, and we discuss how further aspects can be easily introduced if required.

Keywords: Wireless Sensor Networks, Intrusion Detection Systems, Security-Cost Tradeoffs,

1. Introduction

Wireless Sensor Networks (WSNs) consists of a potentially large number of devices with sensing capabilities and wireless communication links. WSNs have recently proliferated thanks to their relatively inexpensive cost and their application in domains such as monitoring of physical phenomena or medical applications.

Security concerns have been at the core of research activities in WSNs almost since the earliest discussions on these technologies. One aspect that have received some attention is the possibility of deploying Intrusion Detection Systems in such environments (see e.g. [1]). The deployment of a single IDS is a practical challenge due in part to the poor understanding of the relationship between the wide range of detection modules and tuning parameters offered by the IDS, and the computational resources (e.g. CPU and memory) consumed by each different combination of these. This aspect acquires some extra relevance in the WSN domain, as in many cases the network will not have the computational resources needed.

Dreger et al [2] have recently presented an approach to automatically derive site-specific IDS configurations given predefined resource constraints. Motivation for this work can be found in the original paper, where the authors provide sound arguments regarding how resource exhaustion affects the quality of network monitoring, ultimately becoming a significant parameter to be taken into account. Dreger et al’s model does not incorporate a notion of “detection quality”. Rather, it focuses on which types of analyses are feasible under given resource constraints. It is implicitly assumed that the more the number of analysers deployed, the more the detection quality. Even assuming that this is true, it is not clear that in a distributed IDS the optimal deployment of sensors ("optimal" both in terms of detection coverage and resource consumption) corresponds to the optimal configuration of each individual node.

The main contribution of this work is a formal model to better understand the intrinsic complexity of placing and configuring detection nodes optimally. In particular, we show that the PCP is NP-complete in the general case. To an extent this is an unsurprising result, as the problem belongs to a wider and well-known class with similar properties. This notwithstanding, we believe our model serves to clarify the main elements present in the problem and their characteristics. Furthermore, we also show how heuristic search algorithms can be successfully applied to find near-optimal (or, simply, good-enough) solutions.

The rest of the paper is organised as follows. In Section 2 we discuss some previous work related to the placement and/or configuration of detection infrastructures. Our for-
nal model is described in Section 3, and Section 4 provides a description of the problem and proof of its hardness. In Section 5 we introduce a heuristic search approach to solve the PCP and in Section 6 report some experimental results. Finally, Sections 7 and 8 summarise our main contributions and discuss future work.

2. Related work

Placement problems in traditional networks have received much attention over the last decades. The problem is known to be both relevant and complex, and several approaches have been proposed to decide where to place the often scarce detection capabilities and, to a lesser extent, how to configure them, particularly in schemes based on node clustering [3]. We next discuss some relevant works in this area.

Noel and Jajodia proposed in [4] the use of attack graphs to find out optimal placement for IDS nodes. An attack graph is a formal tool that captures possible paths taken by potential intruders to attack a given asset. Such graphs are constructed topologically and taking into account both vulnerable services that allow nodes to be exploited and used as launch pads, and protective measures deployed to restrict connectivity. The purpose is to enumerate all paths leading to given assets and, subsequently, choose the optimal placement to monitor all paths using a minimal number of sensors. This is seen as a set cover problem: each node allows for monitoring of certain graph edges and the challenge is to find a minimum set of routers that cover all edges in the graph; a greedy algorithm is then used to compute optimal placement. The use of attack graphs provides an efficient mapping of network vulnerabilities in the network. A vulnerability-driven approach to deploying sensors overlooks factors such as traffic load however. As a result, the placement is optimised such that the more paths that go through a node the more likely it is chosen for placement.

Rolando et al. introduced in [5] a formal logic-based approach to describe networks and automatically analyse them to generate signatures for attack traffic. This approach serves too to determine placement of detection nodes that incorporate such signatures. Their notation to model networks is simple yet expressive to specify network nodes and interconnecting links in relevant detail. A prototype in Prolog is used to infer solutions from given specifications.

Multi-objective optimisation approaches have also been proposed recently [6]. Here the idea is to use heuristic search techniques to explore placements that maximise some goal function, and the feedback is provided through simulation.

To the best of our knowledge, the problem has received much less attention in the case of WSNs. In these scenarios, factors such as resource consumption acquire much more relevance due to the nature of the network and the detection nodes.

3. Formal models

In this section we introduce models for the network, the attacker, the IDS nodes and their resource consumption. These models will serve to provide a precise formulation of the placement-configuration problem.

3.1. Network and traffic models

Definition 1 (Wireless Sensor Network). A WSN is a tuple $\mathcal{W} = (S, L, \mathcal{R}, \text{conn})$, where:

- $S = \{s_1, \ldots, s_{|S|}\}$ is a set of sensors.
- $L = \{l_1, \ldots, l_{|L|}\} \subseteq S \times S$ is a set of communication links between them.
- $\mathcal{R} = [r_{ijk}]$ is a routing function, modelled by an $|S| \times |S| \times |L|$ matrix, with $r_{ijk} = 1$ if messages from $s_i$ to $s_j$ pass over link $l_k$, and $r_{ijk} = 0$ otherwise.
- $\text{conn} : L \times \mathbb{N} \to \mathbb{N}$ is a traffic function such that $\text{conn}(l_i, t)$, also written $\text{conn}_i(t)$, gives the number of active connections crossing link $l_i$ at time $t$.

The pair $(S, L)$ defines the network topology as a graph. Note that modelling a network as a graph does not necessarily imply that such a graph corresponds exactly to the actual network topology. Some abstractions can be done, e.g., grouping some subnetworks or sets of sensors into a single node, modelling elements outside the network as an additional node, and so on.

For simplicity, we deliberately choose a static routing structure. Modelling time-varying routes (e.g., due to mobility or congestion) is straightforward and can be easily incorporated if needed.

Function $\text{conn}()$ provides basic information about traffic load and distribution. Such a traffic profile can be easily obtained by means of network monitoring tools. The main reason for introducing a temporal component into the traffic function is twofold. On the one hand, this will facilitate to use realistic traffic models reflecting the various traffic regimes usually present in a network. On the other hand, the resource consumption model introduced later is intimately related to the traffic load that detection nodes must process. Consequently, the use of traffic aggregates such as, for example, an average, may lead to erroneous resource consumption estimators.

3.2. IDS model

Definition 2 (NIDS). A NIDS is a collection of $n$ identical detection nodes $D = \{d_1, \ldots, d_n\}$, each one consisting of a collection of $m$ analysers $d_i = \{a^1_i, \ldots, a^m_i\}$.\footnote{Assumption made for the sake of clarity. The model can be easily extended to NIDS composed of different nodes.}
This structure is motivated by the architecture of most current IDS technologies. Systems such as Bro, NetSTAT, Snort, etc. are intrinsically modular and consist of an “engine” that invokes several components, each one being responsible for one class of attacks. In other cases such modules can be understood as the set of attack signatures, so the specific subset activated during operation defines the detection capabilities of the node.

Next we introduce our configuration model. We assume that each detection node \( d_i \) can be placed at one network link. In turn, each analyser \( a_j^i \) within detection node \( d_i \) can be activated or deactivated by the operator.

**Definition 3 (NIDS configuration).** A configuration for a NIDS is a \( n \times m \) binary matrix

\[
C = [c_{ij}]
\]

with \( c_{ij} = 1 \) if analyser \( a_j^i \) is activated, and 0 otherwise.

When necessary, we will abuse notation and write \( C \) to denote the configuration of detection node \( d_i \).

**Definition 4 (Placement).** A placement \( \mathcal{P} \) is an injective function from the set of detection nodes \( D \) to the set of network edges \( L \)

\[
\mathcal{P} : D \rightarrow L
\]

such that \( \mathcal{P}(d_i) = l_j \) if detection node \( d_i \) is placed at edge \( l_j \).

### 3.3. Resource consumption model

We summarise here the resource consumption model of intrusion detection nodes presented in [2]. For the purposes of this work, we omit some details which are not relevant in our domain.

For each detection node \( d_i \), the resource consumption of a particular analyser \( a_j^i \) is linear in the number of active connections traversing the node where the sensor is deployed, i.e:

\[
RC_j(\mathcal{P}(d_i), t) = \alpha_j \cdot conn_{\mathcal{P}(d_i)}(t) + \beta_j
\]

(1)

Coefficients \( \alpha_j \) and \( \beta_j \) are fixed for each analyser \( a_j \) and must be determined experimentally. Note that such coefficients are identical for all detection nodes, as we assume them identical. This, again, is trivially extensible to an heterogeneous model. We assume \( \alpha_j \) and \( \beta_j \) are given as an input to our model.

Now given a placement \( \mathcal{P} \) and a configuration \( C \), its global resource consumption is simply the sum of the individual resource consumptions of the activated components:

\[
GRC(\mathcal{P}, C, t) = \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} \cdot RC_j(\mathcal{P}(d_i), t)
\]

\[
= \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij}(\alpha_j \cdot conn_{\mathcal{P}(d_i)}(t) + \beta_j)
\]

In general, \( GRC(\mathcal{P}, C, t) \) varies over time depending on:

1. the changes in the number of active connections crossing the nodes where detection nodes are deployed; and
2. the analysers activated on each one of them. The series of values \( \{GRC(\mathcal{P}, C, t)\}_{t=1}^{T} \) can be processed in various ways, depending on the specific objective function. Thus for example, our goal may be minimising the average resource consumption over time, ensuring that no sensor ever surpasses a maximum consumption level, etc. We formalise this in the following definition.

**Definition 5 (Resource consumption cost function).** A resource consumption cost function for a time interval \( [1, T] \) is a positive map \( \rho : \mathbb{N} \times \cdots \times \mathbb{N} \rightarrow \mathbb{R}^+ \) measuring the cost of the consumption series \( \rho(GRC(\mathcal{P}, C, 1), \ldots, GRC(\mathcal{P}, C, T)) \).

### 3.4. Attack model

We now present our attack model. We assume the attack space consists of \( m \) different types (or classes) \( \{att_1, \ldots, att_m\} \), each one associated to the set of analysers described above. Each \( att_j \) should not be viewed as an specific attack instance, but rather as a generic class of attacks potentially detectable by analyser \( a_j \), and only by \( a_j \). This is a completeness assumption, i.e., every attack might be detected if there is a detection node in the path between the attacker and the target and the associated analyser is activated. This, however, does not necessarily imply that correct detection occurs with probability 1. The detection model discussed later incorporates a notion of “detection quality” that takes into account possible imperfections in the detection procedure.

Our attack model is given by a set of probabilistic estimators \( A(s_i, s_j, att_k) \), one for each pair of network sensors \((s_i, s_j)\) and each attack type \( att_k \). The value \( A(v_i, v_j, att_k) \) provides the probability that node \( v_i \) launches an attack of type \( att_k \) against node \( v_j \). The values given by \( A \) do not necessarily define a probability distribution, neither over the set of nodes nor over the set of attacks. In the general case, \( k^2m \) of such estimators are required, though in practice many of them are likely to be null.

It is worth recalling here the remark given in Definition 1 with respect to our network model: every network node \( s_i \) in the network model does not necessarily correspond to a real sensor. Thus, external attackers can be easily modelled through additional nodes from which attacks are executed.

The construction of an attack model of this form can be partially automated through the use of risk assessment tools, but some additional input has to be provided by the security administrator. This can be done in a number of ways, e.g. by relying on past experiences of security incidents or simply by assuming some threat model. Moreover, the model can be dynamically adapted using the information provided by already placed detectors. Thus, the
actual placement would play an important role in providing feedback to the decision making centre for the next update of the placement.

3.5. Detection model

Suppose that a detection node $d_i$ is placed at link $l_j$, i.e. $\mathbb{P}(d_i) = l_j$, and that $c_{ij} = 1$, so analyser $a_j$ is activated. In an oversimplified detection model, this could mean that all attacks of type $att_j$ traversing link $l_j$ would be successfully detected by $d_i$. This, however, is an unrealistic assumption and does not reflect the reality of current detection technologies. In practice, the detection quality of an analyser $a_j$ is characterised by a pair of the form $(TPR_j, FPR_j)$ providing estimates of the true and false positive rates, respectively. We assume the standard interpretation to state to formulate the placement-configuration problem in precise terms. The only remaining piece is a mechanism to rate to formulate the placement-configuration problem in

4. The Placement-Configuration Problem (PCP)

The elements introduced in the previous section facilitate to formulate the placement-configuration problem in precise terms. The only remaining piece is a mechanism to measure how good a particular placement-configuration is. We will formalise this through the notion of a simulation.

**Definition 6 (PC simulation).** Given:

- a network $W$;
- an associated attack model $A$;
- a set of detection sensors $D$; and
- a placement-configuration $(\mathbb{P}, C)$

A simulation of $(W, A, \mathbb{P}, C, \rho)$ is an algorithm $(TP, FP, R) \leftarrow SIM^T(W, A, \mathbb{P}, C, \rho)$ giving the expected number of true and false positives $(FP$ and $TP$, respectively) and the expected resource consumption cost, according to $\rho$, of the placement-configuration $(\mathbb{P}, C)$ over the time period $[1, T]$.

With the model introduced above, it is clear that the main objective is to find both a placement strategy and a configuration for the sensors such that:

(i) the total number of attacks detected $TP$ is maximised;

(ii) the total number of false positives $FP$ is minimised; and

(iii) the resource consumption function $\rho(GRC)$ is minimised.

Various trade-offs can be easily foreseen. For example, the more the number of activated analysers, the more the expected number of detected attacks, at the expense of higher resource consumption. Additionally, putting analysers with high $FPR$ in highly utilised (in terms of traffic) areas of the network would generally increase the number of false positives. We next describe a utility-based approach that takes an economic perspective on this problem.

4.1. Placement-configuration utility

A possible approach to dealing with the conflicts between $TP$, $FP$ and $\rho(GRC)$ is to consider them equally important. We believe, however, that this is generally unrealistic. In practice there may be a complex preference relation between these three goals. For example, the importance of keeping resource consumption down could not necessarily be as relevant as detecting every possible attack. It is also unclear how costly will be to deal with false alarms and, more importantly, how this compares to the other two factors. Facing a trade-off, real world decisions are often made on an economic basis: no matter how good a security technology is, it is worth having for an organisation if and only if its (economic) cost is lower than the loss they incur by not having it. On the same principles, we can easily assume that each one of the three factors considered above has an economic utility for the organisation operating the network, and that the practical value of a given placement-configuration is its overall economic utility. A solution is therefore better than another if it yields greater economic utility, this being measured by some domain-specific function. We next formalise this.

**Definition 7 (PC utility functions).** A placement-configuration utility function $\xi : \mathbb{N} \times \mathbb{N} \times \mathbb{R} \to \mathbb{R}^+$ is a positive map $\xi(TP, FP, R)$ summarising into a single positive real number the utility of a simulation outcome $(TP, FP, R)$.

For example, one of the simplest utility functions is a linear combination of the three simulation outcomes, with the associated weights reflecting their relative importance:

$$
\xi(TP, FR, R) = \omega_{TP} TP + \omega_{FP} FP + \omega_{R} R
$$

4.2. The PCP problem

**Definition 8 (PCP).** Given a WSN $W$ and an attack model $A$, find a placement $\mathbb{P}$ and a configuration $C$ maximising some given utility function $\xi$ of $SIM^T(W, A, \mathbb{P}, C, \rho)$. That is:

$$
\max_{(\mathbb{P}, C)} \xi(SIM^T(W, A, \mathbb{P}, C, \rho))
$$
4.3. NP-completeness of the PCP

We now prove that the decision problem associated with the PCP, denoted PCP$^{\text{dec}}$ is computationally intractable. We proceed by restriction, which, as pointed out in [7], is perhaps the simplest and most frequently applicable proof of NP-completeness\(^3\). We will reduce the PCP to the KNAPSACK problem, which for completeness we next reproduce.

**Definition 9 (KNAPSACK).** Given a finite set \( U \), for each \( u \in U \) a size \( s(u) \in \mathbb{Z}^+ \) and a value \( v(u) \in \mathbb{Z}^+ \), and positive integers \( B \) and \( K \): Is there a subset \( U' \subseteq U \) such that \( \sum_{u \in U'} s(u) \leq B \) and such that \( \sum_{u \in U'} v(u) \geq K \)?

**Theorem 1.** KNAPSACK is NP-complete

**Proof.** See [8].

**Theorem 2.** The PCP$^{\text{dec}}$ is NP-complete.

**Proof.** We first restrict the problem to instances where every detection node has exactly \( m = 1 \) analysers and the configuration matrix is \( C^T = (1 \ 1 \ \cdots \ 1) \), i.e., the analyser is activated in every detection node, so no configuration problem is in reality present.

We first label the network graph given by \( (S, L) \) by associating a pair \((y_i, n_i)\) with each network link \( l_i \in L\). The value \( y_i \) measures the utility obtained should a detection node is placed on \( l_i \). (Recall that no configuration is needed.) Likewise, \( n_i \) corresponds to the opposite situation, i.e., the utility resulting from not having a detection node on \( l_i \). Both values can be estimated offline by running [\( |L| \) simulations with \( n = 1 \) detection nodes placed on every possible link \( l_i \in L \).

We now construct a set \( U \) with exactly \( 2|L| \) elements as follows. Initially \( U = \emptyset \) and 2 elements, \( u^0_i \) and \( u^1_i \) are added for each network link \( l_i \in L \), with:

- \( v(u^0_i) = n_i \) and \( s(u^0_i) = 0 \)
- \( v(u^1_i) = y_i \) and \( s(u^1_i) = 1 \)

The problem now is clearly a KNAPSACK with upper bound for cost \( B = n \) and lower bound for benefit \( K \) equal to the upper bound for utility in the original PCP$^{\text{dec}}$. \( \Box \)

**Remark 1.** It is worth noting that the placement problem itself, with no configuration whatsoever, is NP-complete. In practical terms, the configuration factor will add some extra complexity.

\(^3\) Roughly speaking, a proof by restriction consist of showing that a problem contains a known NP-complete problem as a special case. The general idea is to place additional restrictions so that the resulting problem is (or is trivially equivalent to) a known NP-complete problem.

---

**Algorithm 1: PCP search**

1. \((\bar{P}, \bar{C}) \leftarrow (P, C)\)
2. \(\tau \leftarrow \tau_0\)
3. **repeat until** stopping criterion is met
4. **repeat** MIL times
5. Pick \((P', C') \in_R NEIG(P, C)\)
6. Pick \(U \in_R (0, 1)\)
7. if \(\xi \left(\text{SIM}^U(W, A, P', C', \rho)\right) > \xi \left(\text{SIM}^U(W, A, P, C, \rho)\right) + T \ln U\) then
   \((P, C) \leftarrow (P', C')\)
8. \(\tau \leftarrow \gamma \tau\)

---

**Figure 1:** Basic simulated annealing search algorithm for the PCP problem.

---

**5. Heuristic solutions**

In practical terms, the NP-completeness of the PCP problem is an obstacle to efficiently determining optimal placement-configurations in large WSNs. Over the last decades, problems with a very similar structure have been approached by means of heuristic optimisation procedures [9, 10], in some cases with remarkable results (e.g., [11, 12]). In this section we describe how a well-known local search algorithm can be applied to the PCP problem and report our experimental results.

---

**5.1. Simulated annealing**

Simulated annealing [13] is a search heuristic inspired by the cooling processes of molten metals. Basically, it can be seen as a basic hill-climbing coupled with the probabilistic acceptance of non-improving solutions. This mechanism allows a local search that eventually can escape from local optima.

The search starts at some initial state (solution) \((\bar{P}, \bar{C})\). The algorithm employs a control parameter \(\tau \in \mathbb{R}^+\) known as the temperature. This starts at some positive value \(\tau_0\) and is gradually lowered at each iteration, typically by geometric cooling: \(\tau_{t+1} = \gamma \tau_t, \gamma \in (0, 1)\).

At each temperature, a number MIL (Moves in Inner Loop) of neighbour states are attempted. A candidate state \((P', C')\) in the neighbourhood \(NEIG(P, C)\) of \((P, C)\) is obtained by applying some move function to \((P, C)\). The new state is accepted if its better than \((P, C)\) as measured by the utility function \(\xi()\). To escape from local optima, the technique also accepts candidates which are slightly worse than the current state, meaning that its utility is no more than \(|\tau \ln U|\) lower, with \(U\) a uniform random variable in \((0, 1)\). As \(T\) becomes smaller, this term gets closer to 0, so as the temperature is gradually lowered it becomes harder to accept worse moves.

The algorithm terminates when some stopping criterion is met, usually after a fixed number \(MaxIL\) of inner loops have been executed, or when some maximum
number \textit{MUL} of consecutive inner loops without improvements have been reached. The basic algorithm is shown in Figure 1.

5.2. \textit{Solution representation and search operators}

Each potential solution \((P, C)\) is represented by a vector \(\text{sol}\) of length \(|L|\), where each value \(\text{sol}[i] \in [0, 2^m - 1]\) gives the configuration of a detection node placed at link \(l_i\). Such a configuration is interpreted as an \(m\)-bit number, with the \(k\)-th bit set to 1 if the \(k\)-th analyser is activated and vice versa. A value \(\text{sol}[i] = 0\) is interpreted as not having a detector placed on link \(l_i\). Consequently, a solution vector \(\text{sol}\) is admissible if and only if it contains exactly \(n\) non-null components.

The neighbouring function \(\text{NEIG}()\) is implemented as a random process that generates an admissible solution not too dissimilar from the given one as input. This is implemented by first randomly choosing whether to relocate a detector and keep its current configuration, or else to keep the detector on the same link but with a slightly different configuration. Both processes are equally probable. In the former case, the detector is relocated to a neighbouring link. In our experiments, this strategy has proven most effective. As for changing the configuration of a detector placed at link \(l_j\), it is implemented by flipping the status of a randomly chosen analyser \(a_k\); that is, \(\text{sol}[j] = \text{sol}[j] \oplus 2^k\). This process is repeated until the obtained configuration contains at least one activated analyser.

6. Results

In this section we present some experimental results obtained with an implementation of the search procedure described above and the model introduced in this paper.

6.1. \textit{Experimental setting}

For our experimentation we have relied on simulated WSNs of varying length. The topology is randomly generated according to a power-law degree scheme [14], such as e.g. the case of Barabási and Albert networks [15]. Fig. 2 shows some examples of simulated topologies generated with this procedure.

In terms of number of detection nodes, we define the detection intensity as the ratio between the number of detection nodes and the number of sensors. In doing so, we obtain estimates of the solution quality regardless of the actual network size. In general, the number of detection nodes will be significantly smaller than the number of sensors. Each type of analyser \(a_j\) is associated with a \(\text{FPR}_j\) randomly generated between 0.5\% and 1\%, and a \(\text{TPR}_j\) also randomly generated between 95\% and 98\%. As for the resource consumption model, the values of \((\alpha_j, \beta_j)\) are values randomly generated between 1 and 10. In this last case, the actual interpretation is irrelevant, and the aggregation function \(\rho\) is simply the sum of the resource consumption time series.

The attack model is generated by fixing a fraction of attack nodes. The attack sources and targets are randomly generated, as well as the attack probabilities given by \(A\). In our case we fix the fraction of attacking nodes to 5\% of the network nodes.

Each simulation consists of \(T = 10000\) time steps. At each one of them, every sensor decides whether a new connection has to be established and, if so, sends a message. Attackers behave according to the same principles, and messages are routed over the network. For each link where a detection node is placed, we compute: (i) whether a TP or a FP is generated; and (ii) the amount of resources consumed during this time step. The set of parameters used in the search is shown in Table 1.

6.2. \textit{Solution quality}

We have performed a number of experiments in order to explore the tradeoffs between problem instances with
Table 1: Simulated Annealing parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td></td>
</tr>
<tr>
<td>Max. No. inner loops (MaxIL)</td>
<td>1000</td>
</tr>
<tr>
<td>Moves in inner loop (MIL)</td>
<td>1000</td>
</tr>
<tr>
<td>Max. No. failed inner loops (MUL)</td>
<td>250</td>
</tr>
<tr>
<td>Initial temperature</td>
<td>200</td>
</tr>
<tr>
<td>Cooling rate</td>
<td>0.95</td>
</tr>
<tr>
<td>PRNG</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Move Function</td>
<td></td>
</tr>
<tr>
<td>Prob. moving a detector</td>
<td>0.7</td>
</tr>
<tr>
<td>Prob. reconfiguring a detector</td>
<td>0.3</td>
</tr>
<tr>
<td>Simulation</td>
<td></td>
</tr>
<tr>
<td>Number of simulation steps</td>
<td>1000</td>
</tr>
<tr>
<td>Number of sensors</td>
<td>500</td>
</tr>
<tr>
<td>Fraction of attacking nodes</td>
<td>5%</td>
</tr>
<tr>
<td>$FPR_j$</td>
<td></td>
</tr>
<tr>
<td>$TPR_j$</td>
<td></td>
</tr>
<tr>
<td>$(\alpha_j, \beta_j)$</td>
<td></td>
</tr>
<tr>
<td>Number of analysers per detector $(m)$</td>
<td>5 and 10</td>
</tr>
<tr>
<td>WSN traffic model</td>
<td></td>
</tr>
<tr>
<td>Resource aggregation function $\rho$</td>
<td>Sum</td>
</tr>
</tbody>
</table>

different intrinsic characteristics and the quality of the solutions found. One major parameter is the traffic load that detection nodes must process. We have created three different traffic settings for low, medium, and high traffic load. These are simulated by assuming that each wireless sensor establishes connections with the sink with delays following a Poisson distribution with parameter $\lambda$. In our case, the three traffic regimes are given by $\lambda_L = 1$, $\lambda_M = 5$ and $\lambda_H = 10$. The number of analysers $m$ (or, equivalently, the number of attack classes) plays an important role too as it contributes to increase the size of the search space. We have experimented with $m = 5$ and $m = 10$ analysers per detection node.

Fig. 3 shows the average utility achieved by the best placement-configuration found versus the detection intensity for different traffic regimes and different number of analysers per node. The curves are consistent with our intuition:

- Higher detection intensities lead to solutions with greater utilities.
- The more the number of analysers per detection node, the lower the quality of the solutions found (with the same search effort).
- The traffic load has a severe impact in the search process and the overall solution quality. Depending on the relative importance given to resource consumption in the utility function, the search might well reward solutions that deliberately miss some attacks if the detection nodes are placed on low-traffic network segments. Thus, the decrease in the number of TP is compensated by a lower number of FP (as less traffic is processed) and lower resource consumption. If this is an undesirable effect, then the relative weights of the three simulation outcomes must be re-assessed.

6.3. Search dynamics

Fig. 4 shows the typical behaviour of a search in terms of the evolution of the best solution found so far. The process starts with a randomly chosen solution that generally yields very low utility. During the first few thousands iterations, no significative progress is often made and the candidates explored show similar utilities. After this stage, the search “enters” an area of high-quality solutions and rapidly improves, almost continuously, the best solution found so far. Eventually, no further significative improvement can be done and the remaining iterations are devoted to refining the best placement-configuration.

We believe this behaviour is connected to the inherent non-linearity of the underlying problem and the guidance
properties provided by the utility function. Thus, small changes in a solution (as provided by the \( NEIG() \) function) do not translate into significant changes in the utility of the solutions, hence the little progress done during the first search stages. Once a “sufficiently good” region of the solution is found, the local search mechanism is able to refine it in subsequent moves.

### 6.4. Search consistency

Given that the search procedure is heuristic in nature, finding an optimal solution is not guaranteed at all. In these cases, an important factor is the ability of the search to consistently find good solutions. Fig. 5 shows the distribution of the quality of the best solution found for 1000 searches on the same problem instance with random initialisation. As shown, the algorithm’s outcome is systematically high, though not necessarily optimal. This reinforces the view that the placement-configuration thus obtained is often sufficiently good but possibly admits further refinements. When possible, we suggest running the search as many times as possible and then choosing the best solution among the pool of candidates obtained.

### 7. Discussion

Our planned future work includes replacing static placement strategies by dynamic ones where detection nodes can be both relocated and reconfigured (see e.g. [16] for a motivating scenario). A natural approach is representing dynamic strategies for each node \( d_i \) as a joint distribution over the set of links \( L \) and the node configuration \( C_i \). Thus \( P(d_i, l_j) \) determines the probability of placing \( d_i \) at link \( l_j \), whereas \( C_{ij} \) is interpreted as the probability of activating analyser \( s_j^i \). Such strategies can be periodically sampled to relocate and reconfigure detection nodes across the network. Likewise, moving from a centralised searching scheme to a distributed protocol is an important avenue. However, in such a case the configuration effort would fall on the sensors. Given their limitations in terms of computing power and energy, such protocols will necessarily be very lightweight.

Additionally to this, the use of utility functions such as those suggested in this work comes at a price: it is difficult to understand the actual tradeoffs among the different evaluation goals yielded by simulating a solution. A more appropriate approach is to attack the problem as a multi-objective optimisation one [17]. The main aim would be to approximate the Pareto-optimal front, that is, the set of solutions that are not dominated by any other solution. Such a surface reflects the inherent tradeoffs and relationships among the different goals. Furthermore, the set of solutions in the Pareto front is a valuable outcome, as the decision-maker can establish under which circumstances a different tradeoff is required and, consequently, a different solution.

Finally, in this work we have deliberately omitted studying the connections, if any, among topological properties of the network, of the attack model and of optimal placements-configurations. This information could be useful to provide further guidance to any placement-configuration algorithm.

### 8. Conclusions

In this work we have discussed the problem of how to choose the best locations in a WSN to place intrusion detection nodes, and how to configure them. We have presented a formal model that tries to capture the main factors present in the problem whilst simultaneously avoiding some elements of complexity. With this formulation, we have shown that the PCP problem is NP-complete and we have explored the performance of a well-known search heuristic.

Even though a number of simplifications have been made, we believe that they do not affect the overall validity of our main results. For example:

- Detection nodes of different characteristics can be modelled with very little extra effort.
- Different utilities for different types of attacks and/or different target sensor nodes can be easily introduced, as this only affects the computation of the global utility functions.
- A distributed NIDS incurs a communication cost if detection nodes need to exchange messages among them and/or with some central location. So far we have assumed that such a cost is negligible, but this may not be true in all the environments. Minimising the overall communication cost may be an additional goal.
- Further restrictions, e.g. the resource consumption or the FP never surpassing some limit could be modelled within the appropriate \( \rho \) function.
More importantly, most of our model’s components have no influence on the inherent structure of the PCP. Thus for example, the resource consumption model or the utility functions can be replaced by others more appropriate for some particular scenarios. It should be clear that such changes do not affect the inherent hardness of the problem coming from the combinatorial state explosion.

As a proof of concept, we have explored the performance of heuristic search algorithms to find solutions to the PCP. In particular, our experimentation with Simulated Annealing shows that, at least for manageable problem instances, satisfactory solutions can be found. It remains to be seen whether the quality scales well with the problem size, and how alternative search algorithms perform.

Acknowledgments

This work was supported by the EPSRC grants EP/E028128/1 (System-Smart Intrusion Detection) and EP/J017515/1 (DAASE: Dynamic Adaptive Automated Software Engineering).

References


Authors’ Biographies

Juan E. Tapiador is Associate Professor of Computer Science in the Computer Security Lab (COSEC) at Universidad Carlos III de Madrid. Previously he was Research Associate at the University of York. His main research interests are in applied cryptography and network security. He holds a Ph.D. in Computer Science from the University of Granada. For additional information see: http://www.seg.inf.uc3m.es/~jet

John A. Clark is Professor of Critical Systems at the University of York. His work is focussed on software engineering (particularly testing) and secure systems engineering. Before joining York in 1992 he worked on UK Government-funded evaluation and R&D security projects, and he has provided consultancy to industry on various aspects of dependability modelling. For additional information see: http://www-users.cs.york.ac.uk/jac