Learning attack mechanisms in Wireless Sensor Networks using Markov Decision Processes

Juan Parras*, Santiago Zazo

Information Processing and Telecommunications Center
Universidad Politécnica de Madrid
ETSI Telecomunicación, Av. Complutense 30, 28040 Madrid (Spain)

Abstract

In this work, we identify two related problems that arise in many Wireless Sensor Networks defense mechanisms: the problem of ad-hoc defense and the problem of optimality. These problems open the door to attacks that could severely affect the performance of defense mechanisms. In this work, we use Markov Decision Processes as framework to model an attacker that is able to exploit these two problems. This allows us to model a defense mechanism theoretically and to obtain the performance of an attack against it, as well as to obtain the optimal attack against the defense mechanism - i.e., the attack that harms the most the defense mechanism. We also make use of Deep Reinforcement Learning tools, showing that they can be used by an intelligent attacker to successfully exploit a possibly unknown defense mechanism, providing a compromise between attack results and computational cost. We test our approach by thoroughly studying a Cooperative Spectrum Sensing attack, which we use to illustrate the framework proposed and to highlight their strengths and weaknesses.

Keywords: Markov Decision Process, Cooperative Spectrum Sensing, Dynamic Programming, Q-learning, Deep learning

1. Introduction

Wireless sensor networks (WSN) is a field in which there has been a lot of research in the last years (Yang, 2014), (Rawat et al., 2014), (Ndiaye et al., 2017). One of the key challenges that WSN pose is related to security. On one side, the communication protocols and standards used in WSN include security solutions, but most of them are still at a proof-of-concept level according to (Tomić and McCann, 2017). On the other side, the existing defense mechanisms are addressed to concrete attacks in different setups, such as cooperative spectrum sensing (CSS) (Fragkiadakis et al., 2013), the 802.15.4 MAC protocol (Sokullu et al., 2008) or mechanisms combining different layers (Wang

*Corresponding author
Email addresses: j.parras@upm.es (Juan Parras), santiago.zazo@upm.es (Santiago Zazo)

et al., 2010), to cite some of them. Two related problems that arise with current defense mechanisms present are the problem of ad-hoc defense and the problem of optimality:

- The problem of ad-hoc defense arises because defense mechanisms are designed against concrete attacks and hence, changes in the attack may severely affect the performance of the defense mechanism. Indeed, this is the usual procedure followed in many works: a defense mechanism is shown to be vulnerable to a concrete type of attack and an improved defense mechanism is proposed, as in (Zhu and Seo, 2009), (Noon and Li, 2010) or (Benedetto et al., 2016). This means that a possibly minor attack variation may severely affect the performance of a certain defense mechanism.

- The problem of optimality arises because attack and defense mechanisms are usually complex to model analytically. This means that most defense mechanisms efficiency are empirically evaluated only and hence, we do not know if a concrete defense (or attack) mechanism is optimal against a concrete attack (or defense) mechanism, nor we know how far from optimal performance is that mechanism. Note that this problem is closely related to the ad-hoc defense problem: since we do not know how which is the optimal attack against a certain defense mechanism (i.e., the attack that harms the most the defense mechanism), we do not know how the defense mechanism performs against a variation of the attack.

In this work, we take the perspective of the attacker and show that these two problems can be used by intelligent attackers to exploit defense mechanisms. First, we propose using the framework of Markov Decision Processes (MDPs) to model the attack mechanism in WSN. This allows dealing with the problem of optimality, since MDPs can be optimally solved. But also, using MDPs allows exploiting the ad-hoc defense problem, since an MDP can be learned by the attacker and hence, such an attacker can learn to exploit a possibly unknown defense mechanism simply by interacting with it. We define a procedure, based on MDP, that can be used as a framework to model the attack and defense mechanism in WSN. To the best of our knowledge, it is the first time that this approach is taken. We also illustrate it by means of analyzing thoroughly a CSS example, in order to show the strengths and weaknesses of our proposed approach.

1.1. Prior work

With regards to the problem of ad-hoc defense, as we mentioned before, many works on this topic start by showing how previous defense mechanisms are vulnerable to a concrete attack method and then, propose a new defense mechanism that deals with this concrete attack method. This approach is followed in (Zhu and Seo, 2009), (Min et al., 2009), (Nguyen-Thanh and Koo, 2009), (Yu et al., 2009), (Noon and Li, 2010), (Wang et al., 2010), (Yan et al., 2012) and (Benedetto et al., 2016), to mention only some works in the field of CSS. We will make use of learning to show how these defense mechanisms can be vulnerable to a learning-based attacker.

The problem of optimality is also present in most of these works, which only validate the proposed defense mechanisms via simulations. One exception is (Yan et al., 2012), where the effects of the covert adaptive data injection attack is evaluated on

a distributed consensus-based CSS network. The efficiency and efficacy of their proposed defense mechanism is assessed both analytically and using simulations. However, we focus on a centralized data fusion scheme. Another exception is found in (Kailkhura et al., 2014), where the performance of a centralized CSS scheme under an Spectrum Sensing Data Falsification (SSDF) attack is evaluated. Yet, in their analysis they do not make use of a CSS defense mechanism, which significantly simplifies their analysis: our approach allows taking into account defense mechanisms.

MDPs are a very mature model used in discrete time optimal control problems. They are used to model a situation in which an agent has to choose an action to interact with a stationary environment which is in a certain state; when the agent executes the action, the environment returns the agent a scalar reward and transitions randomly to another state. They can be solved in two different ways. The first is using Dynamic Programming based tools, as shown in (Bertsekas, 1995) or (Thrun et al., 2005). This approach requires explicit knowledge of the transition function of the environment. The second approach does not require explicit knowledge of the transition function and consists in using Reinforcement Learning (RL) tools, as shown in (Sutton and Barto, 1998). Under this paradigm, the agent interacts with the environment, learning to choose actions in such a way that its total reward is maximized. Moreover, recent advantages in the field of deep learning have brought new Deep RL algorithms, such as Deep Q-Networks (DQN) (Mnih et al., 2015) or Deep Recurrent Q-Networks (DRQN) (Hausknecht and Stone, 2015) to name two of them. Deep RL algorithms have significantly outperformed previous RL methods, allowing to cope with high dimensionality problems which previously could not be tackled.

These advances in Deep RL have found a wide variety of applications, from playing Atari games (Mnih et al., 2015) to designing expert systems for financial trading (Jeong and Kim, 2018) and tax evasion behavior (Goumagias et al., 2018). Note also that expert systems have been used previously for security in WSN, as in (Sarigiannidis et al., 2015) or (Wang et al., 2011), without making use of RL tools. There have been several other approaches that have made use of RL tools applied to WSN security problems, such as routing, data latency, path determination, duty cycle management, QoS provisioning or resource management (Alsheikh et al., 2014), to detect spoofing attacks (Xiao et al., 2015), for mobile offloading (Xiao et al., 2016), (Xiao et al., 2017), to avoid jamming (Aref et al., 2017), (Han et al., 2017) and to model DoS attacks (Li et al., 2017). However, none of these works assess the impact of using RL tools in an attacker. Also, to the best of our knowledge, we present the first attacker architecture based on Deep RL.

1.2. Contributions

We summarize here the main contributions of our work:

- First, we propose a unified approach to assess the theoretical performance of a defense mechanism against an attack, by making use of MDP tools. Solving the MDP would provide the optimal attack against a concrete defense mechanism, thus, our approach deals with the problem of optimality. Note that this also deals with the problem of ad-hoc defense: we are able to evaluate how attack variations can affect a defense mechanism. Also, knowing the performance of the optimal

attacker, it is possible to have a worst case bound on the defense mechanism performance. To the best of our knowledge, ours is the first approach with such characteristics.

- Second, we illustrate the approach we propose by thoroughly studying a SSDF attack on a centralized CSS scheme. As defense mechanism, we choose Enhanced Weighted Sequential Zero/One Test (EWSZOT) (Zhu and Seo, 2009). To the best of our knowledge, we develop the first theoretical model of EWSZOT performance, that can be used to assess its performance theoretically under different attacks. We remark that our approach could be extended to other defense mechanism other than EWSZOT.

- Third, to the best of our knowledge, we are the first that apply Deep RL to an attacker. Our approach allows comparing the results of the RL based attack with the optimal attack. Also, using RL tools allows reaching a compromise between the attack results and the computational cost for the attacker.

The rest of the paper is structured as follows: first, we give an introduction to MDPs and explain how to solve them using Dynamic Programming tools, as well as RL tools, where we introduce several RL algorithms. Then, in order to illustrate the framework we propose, we study a CSS attack in a WSN, where EWSZOT is used as defense mechanism. First, we theoretically model the MDP of EWSZOT and solve it using Dynamic Programming tools. This allows evaluating the harm that current proposed attacks can inflict to EWSZOT. We also propose a variant attack that is able to significantly worsen the performance of EWSZOT, due to the problem of ad-hoc defense. We also obtain the optimal attack against EWSZOT, thus, dealing with the problem of optimality. Second, we use RL tools to model an attacker against EWSZOT, highlighting the strengths and weaknesses of this approach. Finally, we assess our approach, draw some conclusions and give several future lines of work.

2. MDP framework

2.1. MDP definition

MDPs are used to model and solve control problems in dynamical systems (Thrun et al., 2005). In these problems, an agent repeatedly interacts with an environment which is in a certain state. After each interaction, the agent receives a reward depending on its action and the system moves to a new state. An MDP is defined as:

Definition 1. *A Markov Decision Process is a 5-tuple $\langle S, A, P, R, \gamma \rangle$, where:*

- *S is a finite set of states s.*
- *A is a finite set of actions a.*
- *$P_a(s^n, s^{n+1})$ is the probability that action a in state s^n and time n will lead to state s^{n+1} in time $n+1$.*
- *$R_a(s^n, s^{n+1})$ is the expected immediate reward received after transitioning from state s^n to state s^{n+1} due to action a.*

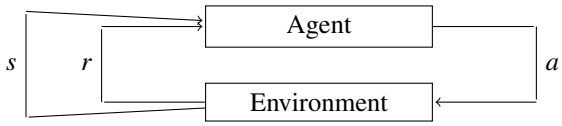


Figure 1: Reinforcement learning basic interaction scheme.

- $\gamma \in [0, 1]$ *is a discount factor.*

A key feature of MDPs is the Markovian property: the probability to reach state s in time n depends only on the previous state in time $n - 1$. In general, MDP can be of finite or infinite horizon, depending on whether the final time N is finite of infinite.

A policy π is a mapping $\pi : S \rightarrow A$ that associates an action (or a probability distribution over actions) to each state. We can evaluate the performance of a policy π in order to obtain its total cumulative expected reward, i.e., the total expected reward obtained by following policy π. We can also find the optimal policy π^*, which is the policy which provides the highest possible cumulative reward. We consider an MDP solved when we have the optimal policy and its cumulative reward. Finally, note that we assume that the state s is fully observable for the agent and that there is only one agent trying to improve its policy.

2.2. MDP solving via DP

Dynamic Programming (DP) algorithm (Bertsekas, 1995), owed to Bellman, can be used to solve an MDP. The main idea in DP is optimizing backwards: we start with the final states s^N and recursively update a value function V^n that contains the optimal reward for each stage n. Mathematically, for $k = 1, ..., N$:

$$V^{N-k}(s^n) = \max_{a \in A(s)} \left\{ R_a(s^n, s^{n+1}) + \gamma \sum_{s^{n+1}} P_a(s^n, s^{n+1}) V^{N-k+1}(s^{n+1}) \right\} \tag{1}$$

where s^n is a state in stage $n = N - k$ and s^{n+1} a state in stage $n + 1 = N - k + 1$. An initial condition is required, which is $V^N(s^N)$ for all states s^N in stage N. This can be thought of as a final reward: an additional reward if the system ends in a concrete state.

In a finite horizon problem (i.e., $N < \infty$), the optimal policy $\pi^{n,*}$ is non-stationary: it depends on the stage n. This optimal policy $\pi^{n,*}$ is obtained also from (1) as:

$$\pi^{N-k,*}(s^n) = \arg \max_{a \in A(s)} \left\{ R_a(s^n, s^{n+1}) + \gamma \sum_{s^{n+1}} P_a(s^n, s^{n+1}) V^{N-k+1}(s^{n+1}) \right\} \tag{2}$$

2.3. MDP solving via RL

Reinforcement Learning (RL) is an area in the artificial intelligence field, which aims to make an agent learning by interacting with an environment, as illustrated in Figure 1. The agent is in state $s \in S$ and chooses an action, denoted by $a \in A$, which affects the environment. The environment provides the agent a numerical reward r and a new state. The objective of the agent is to maximize the total numerical reward

obtained on its interaction with the environment. Using the MDP notation introduced before, RL approximates the optimal policy without knowing the transition probabilities $P_a(s^n, s^{n+1})$. A very complete introduction to the field is given in (Sutton and Barto, 1998).

2.4. RL with Q-learning

Q-learning is one of the possible algorithms that can be used to implement RL (Sutton and Barto, 1998). We again use the concept of value function $V_\pi(s^n)$: it is a mapping that returns the expected payoff obtained in state s^n by using the policy π as:

$$V_\pi(s^n) = R_{a\sim\pi}(s^n, s^{n+1}) + \gamma \sum_{s^{n+1}} P_{a\sim\pi}(s^n, s^{n+1}) V_\pi(s^{n+1}) \tag{3}$$

The state-action value function $Q_\pi(s^n, a)$ represents the expected payoff that the agent would obtain if it is in state s^n and the agent takes action a and follows policy π afterwards as:

$$Q_\pi(s^n, a) = R_a(s^n, s^{n+1}) + \gamma \sum_{s^{n+1}} P_a(s^n, s^{n+1}) V_\pi(s^{n+1}) \tag{4}$$

and observe that both value functions are related as: $V_\pi(s^n) = Q_\pi(s^n, \pi(s^n))$ when policy π is deterministic.

Q-learning algorithm starts by initializing the Q-function in all the $S \times A$ space: we initialize $Q(s^n, a) = 0, \forall s^n \in S, \forall a \in A$. Then, the agent starts to repeatedly interact with its environment: it observes its current state s^n and takes action a following an ϵ-greedy policy, in which the agent chooses to take the action that maximizes $Q(s^n, a)$ with probability $1 - \epsilon$ and with probability ϵ, it takes a random action. Observe that the values of ϵ will regulate the exploration-exploitation trade-off: high values of ϵ will cause that the agent explores the payoff that different actions give him, whereas low values of ϵ will cause the agent to exploit the action that gives him the highest payoff. This causes that the Q-function learned is particularized for this concrete policy: thus we drop the subscript π from the Q-function. When the agent takes action a, the environment transitions to a new state s^{n+1} and it returns an immediate reward r to the agent. This allows updating the Q-function value for state s^n and action a as follows:

$$Q(s^n, a) = Q(s^n, a) + \alpha \left(r + \gamma \max_{a'} Q(s^{n+1}, a') - Q(s^n, a) \right) \tag{5}$$

where $\alpha \in [0, 1]$ is a parameter that controls the learning rate: low values of α means a low Q update, but high values of α introduce a high variance on Q values. This algorithm converges in the limit to the actual Q-function under some mild convergence conditions (Sutton and Barto, 1998). Q-learning algorithm is summarized in Algorithm 1 for the n_r iterations of the algorithm using ϵ-greedy policy.

2.5. RL with neural networks

2.5.1. DQN

A drawback of Q-learning comes when the $S \times A$ space is large: the memory required to store the Q-function can be prohibitively big. This can be overcome by

Algorithm 1 Q-learning algorithm

Input: S,A, n_r
Initialize $Q(s^n, a) = 0, \forall a \in A, \forall s^n \in S$
for n_r repetitions **do**
  Initialize $n = 0$
  Set initial state s^0
  while State s^n is not final **do**
    Obtain action a using ϵ-greedy policy
    Take action a and obtain s^{n+1} and r
    Update $Q(s^n, a)$ using (5)
    Set $n = n + 1$
    Use s^{n+1} for the next iteration
for $s^n \in S$ **do**
  $\hat{\pi}(s^n)^* = \arg\max_a Q(s^n, a)$
Output: $\hat{\pi}(s^n)^*$

using an approximation of the Q-function. A powerful function approximator used is based in Deep Q-Networks (DQN) (Mnih et al., 2013) (Mnih et al., 2015).

DQN approach is based on Q-learning, thus it is similar to Algorithm 1. But under DQN, the Q-function is replaced by a deep neural network, whose input is the current state s^n and its output is the approximated Q-value function $Q(s^n, a)$. Also, DQN uses experience replay to learn. Under this paradigm, in each time, the experience of the agent $e^n = (s^n, a^n, r^n, s^{n+1})$ is stored in a data set E. The set E is updated as new actions are taken by the agents and the data contained in E is used to update the neural network multiple times. Experience replay allows a greater data efficiency, allows avoiding the correlation between consecutive samples and it helps avoiding oscillations or divergence in the network (Mnih et al., 2013).

Another feature of DQN consists in using target networks: at the beginning of a training epoch, the neural network is cloned and the copy is called target network. The target network is used to update the neural network during that epoch: the Q-value function values for future rewards are obtained from the target network. Because target networks are updated only once per epoch, that gives better convergence results (Mnih et al., 2015).

2.5.2. DRQN

After DNQ came out, there were several variations on this concept. A very interesting one is based on Deep Recurrent Q-Networks (Hausknecht and Stone, 2015). It consists in using DQN algorithm with a recurrent neural network to approximate the value function. A recurrent neural network (RNN) is a network specially designed to process sequential data (Goodfellow et al., 2016). An RNN feedbacks its output in time n to a variable called hidden state, which then is used to process the output at time $n + 1$. Thus, RNNs are able to store information about the past inputs on this hidden state, which can be thought of as a feedback loop.

The main problem with RNNs is that these networks are hard to train. There has been a significant effort addressed to alleviate this (Sutskever, 2013). Nowadays, the

main architecture used to implement a RNN is the LSTM (long-short term memory), owed to Hochreiter (Hochreiter and Schmidhuber, 1997). Other structures, such as GRU, have been proposed recently, yet their advantages over LSTM are not clear (Greff et al., 2017). The original DRQN article, indeed, used LSTM (Hausknecht and Stone, 2015). However, RNNs are specially designed to deal with sequences and thus, they are a good option for control problems. Indeed, they could even manage Partially Observable MDP (Hausknecht and Stone, 2015).

3. Problem description

3.1. Problem setup

In order to illustrate the advantages of the MDP framework, we thoroughly study a SSDF attack in a CSS WSN. Note that there are two main sources of error in a CSS scheme: the probability of error of the sensing mechanism and the the presence of malicious sensors. Under a spectrum sensing data falsification (SSDF) or byzantine attack, there are ASs in the network which provide a false report to the FC (Fragkiadakis et al., 2013). Several defense mechanisms have been proposed to deal with SSDF attacks, depending on whether the sensor sends only their decision to the FC (hard fusion) or it also includes additional information about the certainty of the decision of the sensor (soft fusion). Some defense mechanisms for the hard fusion case are Weighted Sequential Probability Ratio Test (WSPRT) and EWSZOT (Zhu and Seo, 2009) and for the soft fusion case, Enhanced WSPRT (EWSPRT) (Zhu and Seo, 2009) and a statistical test based on the energy level distribution (Wang et al., 2010). Many challenges remain open in this field, as (Zhang et al., 2015) shows.

We consider that we have a WSN with M sensors: M_g is the number of good sensors and M_a the number of ASs (where $M = M_g + M_a$). The network uses EWSZOT as defense mechanism. Each report from the sensors to the FC is denoted by the binary variable u_m, where $u = 1$ means that the channel is busy, $u = 0$ means that the channel is idle and m indexes the sensors ($m \in \{1, 2, ..., M\}$). We do not consider the details of the sensing mechanism used by the sensors (see (Zhang et al., 2015) for some possible schemes). Each of the sensors can make a wrong sensing decision with probability P_c. We consider P_c to be constant and independent among the sensors. Thus, in this case, each u_m follows a Bernoulli distribution of parameter P_c if $u = 0$ and $1 - P_c$ if $u = 1$. We base our theoretical model in P_c because it simplifies our analysis significantly and it is flexible enough to take into account different phenomena: related to the channel as the shadowing and fading in the sensors or related to the sensing procedure chosen. The optimality criterion we use is the total probability of error $p_{e,t}$ in the FC, that is, the probability that the FC decides that the channel is busy when it is idle and vice-versa. Thus, the ASs will try to maximize $p_{e,t}$. Since EWSZOT is a defense scheme based on reputations, the ASs need to attack and also keep a sufficiently high reputation. In other words, maximizing $p_{e,t}$ will require the ASs to camouflage.

We choose EWSZOT because it has a higher performance than other schemes based on sequential tests (such as WSPRT and EWSPRT, (Zhu and Seo, 2009)). It is simple to implement, fast in deciding (Zhu and Seo, 2009) and energy efficient (Cichoń et al., 2016). It is one of the most advanced centralized data fusion schemes against SSDF

attacks (see (Fragkiadakis et al., 2013), (Zhang et al., 2015), (Li et al., 2014) or (Liu et al., 2011)).

3.2. EWSZOT description

EWSZOT data fusion scheme is a hard fusion scheme based on reputations. We call stage to a hypothesis test (HT). A HT consists in the FC asking reports to certain sensors and taking a decision based on these reports. In each stage n, the reputation of the sensor m, r_m^n, is updated based on whether the report of sensor m was consistent or not with the decision taken u_d^n (we use superscripts to denote the stage). Mathematically:

$$r_m^n = \begin{cases} r_m^{n-1} + 1 & \text{if} \quad u_m^n = u_d^n \\ r_m^{n-1} - 1 & \text{if} \quad u_m^n \neq u_d^n \end{cases} \tag{6}$$

In stage $n = 1$, all reputations are initialized to 0. The decision rule used by EWSZOT HT is:

$$\begin{cases} u_d^n = 1 & \text{if} \quad W^n \geq q \\ u_d^n = 0 & \text{if} \quad W^n \leq -q \\ u_d^n = 1 & \text{if} \quad -q < W^n < q \text{ and } m = M_m \\ \text{take another round} & \text{if} \quad -q < W^n < q \text{ and } m < M_m \end{cases} \tag{7}$$

where q and M_m are predefined thresholds. Observe that the first three conditions from (7) are the final conditions: they finish the HT and lead to a decision. Also, W^n is the HT statistic, following:

$$W^n = \sum_{i=1} (-1)^{u_i^n + 1} w_i^n \tag{8}$$

where w_i^n are weights that will be defined later.

EWSZOT HT is similar to a sequential test. The decision rule in stage n consists in asking sensor i to give a report: if its report is $u_i^n = 0$, then W_n is decreased w_i^n units, and if its report is $u_i^n = 1$, then W_n is increased w_i^n units. This process is repeated until:

1. W^n surpasses a threshold q. In this case, the decision is immediately taken using the first two lines of (7).
2. M_m sensors have been called and W^n has not surpassed the threshold q. This means that the test result is uncertain and we follow a conservative decision rule: to decide that the channel is occupied. This decision benefits PUs. This test truncation is added in (Zhu and Seo, 2009) in order to avoid deadlocks.

Finally, the reputation of each sensor has an impact on the HT through the weights w_i^n, defined as:

$$w_m^n = \begin{cases} 0 & \text{if} \quad r_m^n < -g \\ \frac{r_m^n + g}{avg(r_m^n) + g} & \text{if} \quad r_m^n \geq -g \end{cases} \tag{9}$$

where $avg(r_m^n)$ is the average reputation of all sensors and g is a small positive value. The purpose of the weights scheme (9) is that sensors with better reputation have a higher influence on the HT. The use of g allows good sensors to have a slightly negative reputation, caused by their sensing error P_c.

Algorithm 2 EWSZOT algorithm implementation

Input: M_m, q, g

Initialize $r_m^1 = 0, \forall m$
for Stages $n = 1, 2, ...$ **do**
  Obtain weights using (9)
  Select the M_m sensors with highest reputations
  for Sensors selected **do**
    Ask report from sensor
    Update W^n using (8)
    if $W^n \geq q$ or $W^n \leq -q$ or M_m sensors have been called **then**
      Take decision u_d^m using (7)
      Exit loop
  Update reputations using (6)

Reputations also determine the order in which sensors are asked to give their reports. EWSZOT calls up to M_m sensors in descending order of reputations. Thus, we ensure using the sensors with best reputations to take the decision. The whole procedure is summarized in Algorithm 2, where an implementation of EWSZOT is presented.

3.3. Attacks against EWSZOT

We use three different attack strategies:

- Dummy strategies. These strategies consists in always doing a predefined action. Even though they are basic attacks, they are widely used. In (Zhu and Seo, 2009), it is shown that EWSZOT is successful against three possible attacks: report the channel always busy, always idle or always give a false report. To the best of our knowledge, these are the current attacks against EWSZOT. Note that the problem of optimality arises here: no study about the optimality of these attacks is found in (Zhu and Seo, 2009).

- Optimal strategies. These strategies are the result of modeling the defense mechanism and optimizing. In our framework, it means that the strategy is obtained by optimizing an MDP. They are theoretically optimal, but also may be complex to obtain. Note that these strategies may exploit the problem of ad-hoc defense: as we will see, minor changes in the attack may lead to a dramatic performance drop in EWSZOT defense mechanism.

- RL strategies. These strategies are obtained using RL tools, and present an interesting trade-off: they are not as complex to obtain as optimal strategies and provide quasi-optimal results. To the best of our knowledge, this is the first work in which these strategies are compared to the theoretically optimal strategies.

We further illustrate the problem of ad-hoc defense by noting that EWSZOT does not take into account the possibility of an attack addressed to the communication mechanism between the sensors and the FC. We define two different attack situations: a

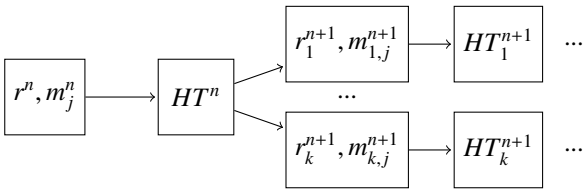


Figure 2: EWSZOT algorithm modelling illustration. Each HT receives as input a reputation vector and a number of jammed sensors and produces a certain number k of updated reputation vectors and jammed sensors. These vectors are used as inputs to new tests in next stages. Each HT has as many k outputs as leaves. Each HT procedure is found using Algorithm 4.

standard SSDF attack (SA), consisting in sending false reports to the FC. Note that SA is the always false attack proposed in (Zhu and Seo, 2009), which will serve as baseline to compare our results with.

The second attack situation we consider is a novel, combined attack (CA), consisting on a SSDF attack and also, a jamming attack addressed to the communications link. When the FC asks for reports to the sensors, these reports may not arrive on time or arrive corrupted, due to channel problems, such as shadowing or fading. But the ASs could also jam the communication link to cause the same effect on the reports sent by good sensors. A survey on different jamming techniques can be found in (Mpitziopoulos et al., 2009). For instance, if a narrowband communication scheme is used, an AS can transmit noise in order to cause a high interference that makes the communication between the sensor and the FC impossible. Or if a CSMA/CA access scheme is used, an AS can jam the communication between a sensor and the FC by simply sending bursts of noise in the backoff periods (Sampath et al., 2007). In order to keep our scheme as simple and broad as possible, we will consider that the group of ASs can jam up to M_j good sensors. When the FC does not obtain a report from sensor m, considers that $u_m = 1$, to be consistent with the test truncation used in EWSZOT HT. . Observe that in our current framework, we consider for simplicity that all corrupted reports come from jamming, but jamming causes actually only part of them.

4. Modeling EWSZOT using an MDP

4.1. States definition

We use as state the tuple formed by the reputation vectors of good and attacking sensors, r_g and r_a respectively; and the number of sensors already jammed in case of CA, m_j. Thus, in stage n, the state is the tuple $s^n = \langle r_g^n, r_a^n, m_j^n \rangle$. The initial state s^0 will be always a vector in which all reputations are set to 0 because that is the initialization value of EWSZOT and $m_j^0 = 0$ because there is no sensor jammed yet.

4.2. Actions definition

The definition of the action space A is subtle. The possible actions of the ASs in case of CA depend on the state due to the jamming. If we have in state s^n that the ASs have already jammed m_j^n sensors, where $m_j^n \leq M_j$, they can only jam up to $M_j - m_j^n$ sensors more. Thus, observe that the action set depends on the state. Note also that if

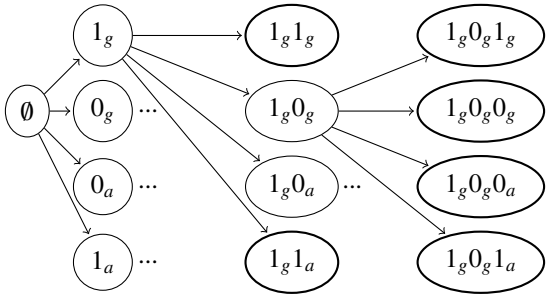


Figure 3: Illustration of EWSZOT HT tree. Each node contains the sequence of reports. For simplicity, we plot part of the tree when $M_m = 3$. Leaves are the thicker nodes. Observe that the leaves may happen when any of the final conditions from (7) is satisfied.

several sensors have the same reputation, the ASs do not know in advance which sensor will be called by the FC. In order to overcome these problems, we define two vector of actions, a_g and a_a. The first is a vector of length M_m which contains the actions for the case that good sensors are called. Each entry of this vector can have two values: 1 if there is jamming and 0 if there is no jamming. Observe that $\sum a_g \leq M_j - m_j^n$, that is, there is a limit on the maximum number of good sensors that could be jammed. We limit the length of this vector to M_m since this is the maximum number of sensors called by the FC: in the extreme case that all sensors called are good, only actions from this vector are used.

The second vector, a_a, is a vector of length $\min\{M_m, M_a\}$ which contains the actions for the case that attacking sensors are called. Each entry of this vector can have two values: 1 if the sensor gives a false report (attack) and 0 if the sensor gives a true report. Note that the limit in length responds to the case in which $M_a < M_m$, that is, there are less ASs than sensors that can be called in each stage. The action space A is formed by all possible combination of action vectors $a = \langle a_g a_a \rangle \in A$. The dimensionality of A is upper bounded by $2^{2M_m} = 4^{M_m}$, which is the maximum number of actions available to the ASs.

In case of SA, there is no possibility of jamming. Thus $a = a_a$, and hence, the dimensionality of A is upper bounded by 2^{M_m}. Note that in SA, the set of actions does not depend on the state.

4.3. Transition probabilities definition

Given a state s^n and an action a, now we turn to compute $P_a(s^n, s^{n+1})$, that is, the probability of transitioning from state s^n to state s^{n+1} due to action a. This requires to model the HT from the EWSZOT mechanism (see Figure 2).

We model the HT in stage n using a tree. Each node of the tree represents the possible sequence *seq* of reports that the FC receives from the sensors. Hence, each parent node will have four children nodes, because it can receive as report $u_m = 0$ from a good sensor (0_g) or from an AS (0_a), $u_m = 1$ from a good sensor (1_g) or from an AS (1_a). The maximum length of the sequence *seq* - and hence, the maximum depth of the tree - will be M_m. Also, observe that a sequence *seq* that satisfies any of the final

conditions from (7) becomes a leaf: it will have no children nodes. An illustration is in Figure 3.

Each node of the tree will store the following data: the sequence of reports received *seq*, the probability of the sequence p_s, the updated number of sensors jammed m_j^{n+1} ($m_j^{n+1} \leq M_j$) and the W^n value. We require as inputs r_g^n, the reputations of the good sensors; r_a^n, the reputations of the ASs; the number of sensors already jammed m_j^n; $p_{1,g}$ ($p_{1,a}$), the probability that a good (attacking) sensor reports $u_m = 1$ (observe that these values depend on P_c), the maximum number of sensors that can be jammed M_j; and the parameters g, M_m and q. Note that the first three parameters are the state: $s^n = \langle r_g^n, r_a^n, m_j^n \rangle$.

We then obtain all the nodes and the data in each node as follows. First, we order the sensors by reputations using r_a^n and r_g^n and obtain $p_g(r)$: the proportion of good sensors among all sensors with reputation value r. Then, we proceed to build the tree. We initialize the root node with $seq = \emptyset$, $p_s = 1$, $m_j^{n+1} = m_j^n$ and $W^n = 0$. Then, we call sensors in descending order of reputations r. We build four children nodes for each parent, each children with a sequence which is the concatenation of the sequence of the parent node and each of the reports that can be obtained (1_g, 1_a, 0_g or 0_a).

Algorithm 3 p_s updating procedure

Input: seq, $a = \langle a_g, a_a \rangle$, m_j^{n+1}, $report$, W^n, $p_{1,m}$, $p_{1,g}$
1: Obtain na, number of attacking sensors already called, from seq
2: Obtain ng, number of good sensors already called, from seq
3: **if** $report$ comes from a good sensor **then**
4: **if** $report = 1_g$ **then**
5: Update $p_s = p_s p_{1,g} p_g(r)$
6: **else if** $report = 0_g$ **then**
7: Update $p_s = p_s(1 - p_{1,g}) p_g(r)$
8: **if** The action indicates jamming: $a_g(n_g) = 1$ **then**
9: Set $report = 1_g$
10: Update $m_j^{n+1} = m_j^{n+1} + 1$
11: **else if** $report$ comes from an attacking sensor **then**
12: **if** The action indicates attack: $a_a(n_a) = 1$ **then**
13: **if** $report = 1_a$ **then**
14: Update $p_s = p_s p_{1,m}(1 - p_g(r))$
15: **else if** $report = 0_a$ **then**
16: Update $p_s = p_s(1 - p_{1,m})(1 - p_g(r))$
17: **else if** The action indicates no attack: $a_a(n_a) = 0$ **then**
18: **if** $report = 1_a$ **then**
19: Update $p_s = p_s p_{1,g}(1 - p_g(r))$
20: **else if** $report = 0_a$ **then**
21: Update $p_s = p_s(1 - p_{1,g})(1 - p_g(r))$
22: Update $seq = \{seq, report\}$
23: Update $W^n = W^n + (-1)^{report+1} w_i^n$ (equation (8))
Output: p_s, m_j^{n+1}, seq, W^n

The updating procedure of p_s is detailed in Algorithm 3. We first need to obtain na and ng, the number of good and attacking sensors already called. This is done by simply looking at the seq vector, which stores the sequence of reports. Then, if the report we have obtained comes from a good sensor, we update p_s using $p_{1,g}$ and $p_{g,r}$, and then check the action for the $ng + 1$ good sensor: if it indicates jamming, we change the report value to $rep = 1_g$ and update the number of sensors jammed, m_j^{n+1}. We first update p_s in order to obtain the total probability of the sequence seq; if there is jamming, the report is changed to 1 for the update of the HT statistic W^n. If the report came from an AS, and the action for the $na + 1$ attacking sensor indicates to attack, we update p_s using $p_{1,m}$ and $1 - p_{g,r}$. If the report came from an AS but the $na + 1$ action from a_a indicates not to attack, then we update this sensor using the error probabilities of a good sensor ($p_{1,g}$ instead of $p_{1,a}$). Finally, we update the HT statistic W^n using (8).

After a report has been received and p_s and W^n have been updated using Algorithm 3, we check whether the node satisfies any of the final conditions from (7). If it does, then the node becomes a leaf, otherwise, we set this node as father and repeat the procedure.

Finally, we obtain the information from the leaves. First, we obtain the probability that the test ends with result u_d^n simply by adding the p_s of the leaves that satisfy each of the decision conditions from (7). This leads us to obtain $p_{t,1}$ (the probability that the test ends because $W^n \geq q$), $p_{t,0}$ (the probability that the test ends because $W^n \leq -q$), $p_{t,nd}$ (the probability that the test ends because $m = M_m$) and m_j^{n+1}, the updated number of sensors jammed. Second, we update the reputations using (6). Observe that there will be as many updated reputation vectors as leaves. Also observe that the probability of each of these reputations vectors is precisely p_s of the leaf. The whole procedure is summarized in Algorithm 4. Note that Algorithm 4 models each iteration of the outer for loop from Algorithm 2.

Each parent node can have up to four children nodes, one per each possible report that can be received. Some nodes may have no children (i.e., the leaves) or less than four (e.g., all the ASs have already been called and hence $p_{g,r} = 1$). In the worst case, which is that all parents have four children, there will be up to 4^{M_m} possible s^{n+1} states in case of SA. In case of CA, we must take into account that there are $M_j + 1$ possible values of m_j^{n+1}, and hence, in the worst case, there will be up to $4^{M_m}(M_j + 1)$ possible s^{n+1} states. We denote this value by k: $k \leq 4^{M_m}(M_j + 1)$ for CA and $k \leq 4^{M_m}$ for SA.

Using Algorithm 4 to model the HT allows obtaining the exact values of $P_a(s^n, s^{n+1})$: for each possible combination of action a and state s^n, we can obtain the probability p_s of transitioning to state s^{n+1}. Note that the Markovian property is satisfied: each HT test output depends only on its input.

4.4. Reward definition

We can obtain the expected reward $R_a(s^n, s^{n+1})$ at the same time as we obtain $P_a(s^n, s^{n+1})$ using the output of Algorithm 4 as:

$$R_a(s^n, s^{n+1}) = \begin{cases} \frac{1}{N} \sum_{s^{n+1} \in S^{n+1}} p_s(p_{t,1} + p_{t,nd}) & \text{if} \quad u^n = 0 \\ \frac{1}{N} \sum_{s^{n+1} \in S^{n+1}} p_s p_{t,0} & \text{if} \quad u^n = 1 \end{cases} \tag{10}$$

Algorithm 4 EWSZOT HT modelling in stage n for the MDP

Input: $s^n = \langle r_g^n, r_a^n, m_j^n \rangle$, $a = \langle a_g, a_a \rangle$, $p_{1,g}$, $p_{1,a}$, M_m, M_j, q, g

Obtain weights w using (9)
Select the M_m sensors with highest reputations
Initialize tree root: $p_s = 1$, $W^n = 0$, $m_j^{n+1} = m_j^n$, $seq = \emptyset$
for Each node which is not a leaf **do**
  for Each four possible reports rep: 0_g, 1_g, 0_a, 1_a **do**
    Create children node
    Obtain $p_g(r)$ for the reputation of the current sensor
    Update seq, p_s, m_j^{n+1} and W^n using Algorithm 3
    if $W^n \geq q$ or $W^n \leq -q$ or length of seq is M_m **then**
      Take decision u_d^n using (7)
      Make this node a leaf
Initialize $p_{t,0} = p_{t,1} = p_{t,nd} = 0$
for Each leaf **do**
  Update $p_{t,0}$, $p_{t,1}$ or $p_{t,nd}$
  Update r_g^{n+1} and r_a^{n+1} using (6)

Output: $s^{n+1} = \langle r_g^{n+1}, r_a^{n+1}, m_j^{n+1} \rangle$, $p_{t,0}$, $p_{t,1}$, $p_{t,nd}$, p_s

where the reward is the expected error probability conditioned to being in state s^n and taking action a. For our problem, we consider that the total reward is:

$$r = p_{e,t} = \sum_{n \in N} R_a(s^n, s^{n+1}) \tag{11}$$

where we considered that the discount factor $\gamma = 1$.

We consider that our problem is of finite horizon (i.e., $N < \infty$). This has two important consequences. The first one is that the optimal policy will be non-stationary. This means that the optimal policy in state s^n may differ from the optimal policy in the same state in a different time. However, the states in our setup are related to the reputations and in our simulations, we observed that states appearing more than once in the same simulation was rare. Thus, we will make use of stationary policies. This is related to the second consequence: RL is used to learn stationary policies, thus, we can use RL algorithms without further modifications.

4.5. EWSZOT model complexity

We now proceed to evaluate the complexity of the MDP model proposed. Let us assume that we want to evaluate a policy, that is, obtaining $V_\pi(s^0)$. For each HT, there are $k \leq 4^{M_m}(M_j + 1)$ possible transitions to other states. An HT is performed for each state s^n as long as $n \in [0, N-1]$. And we always start from the same s^0. Thus, as we can observe in Figure 2, to evaluate a policy we must obtain a tree containing all possible states that can be transitioned from s^0 by following policy π. The depth of this tree is $N + 1$ and the maximum number of states that the tree can have is bounded, in

case of CA, by:

$$\sum_{n=0}^{N} k^n = \sum_{n=0}^{N} \left(4^{M_m}(M_j+1)\right)^n \tag{12}$$

And in case of SA, the upper bound is:

$$\sum_{n=0}^{N} k^n = \sum_{n=0}^{N} 4^{nM_m} \tag{13}$$

Thus, the state space, though finite, can be very large. It is possible to speed up the computation by:

- After HT in stage n, we drop out all states s^{n+1} such that $p_s = 0$. That is, we delete all states with a null probability.
- After HT in stage n, we do state aggregation: we merge together all states s^{n+1} that are the same and add their probabilities p_s.
- After each stage n, we truncate the set of states s^{n+1} by preserving only the T states s^{n+1} which have the highest probability p_s. This truncation can reduce significantly the computational cost by fixing a maximum number of states in each stage, but introduces an error in the results. Larger values of T yield a lower error and a higher computational cost.

The procedure to evaluate a concrete policy, with these three improvements, is summed up in Algorithm 5. Observe that we need to know the actual u in order to obtain the decision error values. The total decision error $p_{e,t}$ is $p_{e,t} = p_{t,1} + p_{t,nd}$ when $u = 0$ and $p_{e,t} = p_{t,0}$ when $u = 1$. Also, observe that Algorithm 5 models theoretically the EWSZOT algorithm described by Algorithm 2. And finally, observe that this algorithm allows obtaining the performance of EWSZOT when there is no attack by simply setting $M_j = 0$ and always using actions without attack as policy.

To obtain the optimal policy, we need to take into account that the number of actions available in each state is bounded by 4^{M_m} in case of CA. This means that we would have, for each state s^n, 4^{M_m} possible actions that would cause to transition to a maximum of $k \leq 4^{M_m}(M_j+1)$ possible s^{n+1} states. In this case, the dimensionality of the state-action space is bounded, for CA, by:

$$\sum_{n=0}^{N} \left(4^{M_m}k\right)^n = \sum_{n=0}^{N} \left(4^{2M_m}(M_j+1)\right)^n \tag{14}$$

And for SA, where the number of actions is bounded by 2^{M_m}, the state-action space is bounded by:

$$\sum_{n=0}^{N} \left(2^{M_m}k\right)^n = \sum_{n=0}^{N} \left(2^{M_m}4^{M_m}\right)^n = \sum_{n=0}^{N} 2^{3nM_m} \tag{15}$$

Observe that the dimensionality of the problem is a major problem when it comes to evaluating or searching for an optimal policy. That will be a major limitation in the following sections.

Algorithm 5 EWSZOT algorithm modelling

Input: M_m, M_j, q, g, M, M_g, M_a, P_c, u, N, T, π

Initialize $r_g^0 = 1$ and $r_a^0 = 1$ for all sensors and $m_j^0 = 0$
Initialize $s^0 = \langle r_g^0, r_a^0, m_j^0 \rangle$
Initialize $p_{t,0} = p_{t,1} = p_{t,nd} = 0$
Obtain $p_{1,g}$ and $p_{1,a}$ using P_c
for iterations $n = 0 : N - 1$ **do**
  Initialize $S^{n+1} = \emptyset$
  for Each tuple s^n **do**
    Simulate HT: use Algorithm 4) to update the set of S^{n+1} when policy π is used.
    Erase all s^{n+1} such that $p_s = 0$
    Merge all equal s^{n+1} (state aggregation)
    if $Dim(S^{n+1}) > T$ **then**
      Keep the T tuples $s^{n+1} \in S^{n+1}$ with highest p_s
      Normalize the probabilities p_s of the tuples $s^{n+1} \in S^{n+1}$
  Update $p_{t,0}$, $p_{t,1}$, $p_{t,nd}$

Output: $p_{t,0}$, $p_{t,1}$, $p_{t,nd}$

5. Using the MDP model to evaluate dummy strategies

5.1. Dummy strategies description: AFA and JFA

The MDP model we just described can be used to evaluate the performance of a concrete policy. That is, it allows evaluating the performance of a certain attack. We will evaluate the dummy strategies using the procedure described in Algorithm 5 to obtain their theoretical performance. Recall that dummy strategies consist in always doing a predefined action, which we use as baseline. In our problem, we propose the two following dummy strategies:

- Always false attack (AFA): it is an SA in which the ASs always give false reports to the FC. It is a current attack against EWSZOT as shown in (Zhu and Seo, 2009), together with always reporting the channel busy or idle. Note that these two attacks can also be assessed using our approach.

- Jam and false attack (JFA): it is a CA consisting in attacking as long as it is possible: the ASs always give false reports to the FC and jam the communications for the first M_j good sensors called by the FC. We remark that this is a novel attack, which exploits the problem of ad-hoc defense.

5.2. EWSZOT performance under AFA and JFA

We analyze EWSZOT under both AFA and JFA. We use Algorithm (5), with $M = 10$ sensors, $T = 10^3$, $q = 2$, $M_m = 4$ and $g = 5.51$ (as in (Zhu and Seo, 2009)). We choose small values of q and M_m because the spectrum sensing procedure will be repeated often and thus, it must be fast. Also, this allows the complexity of the EWSZOT modeling not to grow excessively, as shown in Section 4.5. For this and the

rest of simulations in this work, we make use of MatLAB ® and Python based scripts to simulate our environment and the attack results.

We test the influence of $M_a = 1$ AS in the network when $N = 5$ for 51 equispaced values of $P_c \in [0, 0.5]$. The theoretical results can be seen in Figure 4, where they are compared to an empirical simulation of EWSZOT averaged 100 times (the empirical simulation followed Algorithm 2) and also to the optimal strategies (explained in the next section). Observe that our theoretical model from Algorithm 5 corresponds to the actual values. Also, observe that JFA is much more harmful than AFA for the same number of ASs: the ability to jam the communication link significantly degrades the performance of EWSZOT. But this degradation only happens when $u = 0$; when $u = 1$ the attack actually improves the decision error. This is due to the conservative EWSZOT jammed report assignation: a corrupted report is considered to indicate that the channel is busy. Thus, JFA seriously affects the CSS procedure when the channel is idle. In order to overcome the damage done by JFA, the FC could implement a jamming countermeasure which will depend on the jamming technique used (several methods are proposed in (Mpitziopoulos et al., 2009)). We remark the influence of the problem of ad-hoc defense: by performing a minor change in the attacker capabilities, namely, by having jamming capabilities, the attack can significantly degrade the defense mechanism performance.

6. Using the MDP model to obtain optimal strategies

Now, we go a step further and proceed to obtain the optimal solutions to the MDP model presented. We use the DP algorithm from (1) and (2) to obtain the maximum error probability that the attackers can achieve and the optimal policy that they must follow in order to achieve that optimal attack. We initialize $V^N(s^N) = 0$ for all states s^N. By doing that, $V^0(s^0) = p_{e,t}$, that is, the value function in the initial state s^0 is the expected total error probability. The main drawback of DP algorithm for our problem lies in the dimensionality of it, as shown in Section 4.5. We use small values in our simulations in order to alleviate the computational cost.

First, we obtain the theoretical error curves under AFA and JFA using Algorithm 5 with a truncation value $T = 10^3$. We use these values as baseline to compare. Then, we obtain the optimal attack strategies using the DP algorithm from (1) and (2). In order to avoid the dimensionality problem, we use $M_m = 4$, $q = 2$ and $g = 5.51$ for the EWSZOT HT. We consider a CSS network with $M = 10$ sensors, of which $M_a = 1$ (only one attacker sensor) and $M_g = 9$ (there are 9 good sensors). We test for $M_j = \{0, 1, 2\}$ (SA and CA), using 51 equispaced values of P_c in the range $P_c \in [0, 0.5]$ and considering $N = 5$ stages, both when $u = 0$ and $u = 1$. Finally, we test and average 100 empirical implementations of the dummy and optimal strategies in order to validate our approach, using Algorithm 2 to implement EWSZOT defense mechanism.

The results can be observed in Figure 4. Note that optimal strategies always yield the highest error and also, observe that our model predicts correctly the empirical values. When $u = 0$, EWSZOT defense system is severely degraded by the optimal strategy, specially when there is jamming. We also can check that AFA is close to be optimal - it is indeed, for $P_c \geq 0.2$ approximately, but when jamming is available, JFA harm is surpassed by the optimal strategies. Note that the highest differences in the

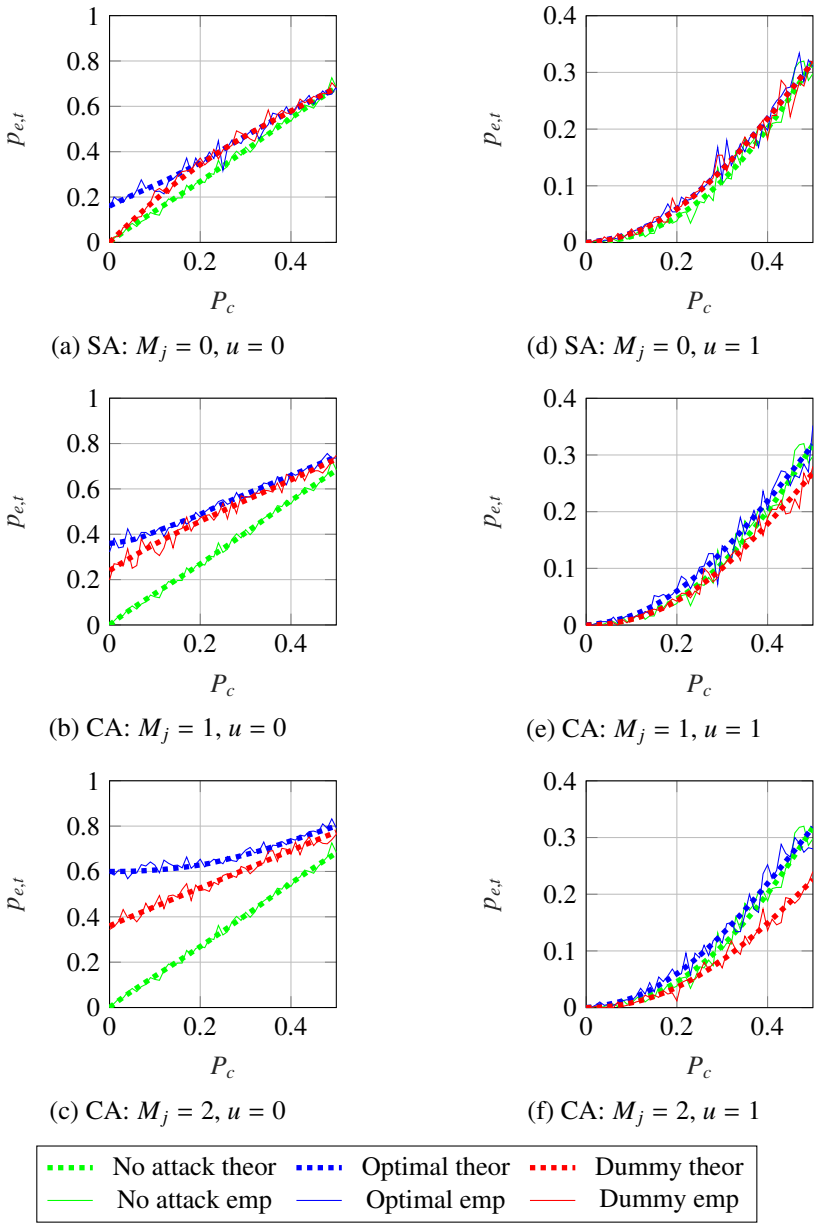


Figure 4: Performance of optimal and dummy attack strategies against EWSZOT in terms of $p_{e,t}$ as a function of P_c.Observe that optimal strategies always yield the highest errors, as expected. Note that dummy strategies are usually not optimal, specially for low P_c values.

attacks happen when there is a low P_c: in systems with a low probability of spectrum sensing error, the optimal attacks can notably degrade the system performance. When

$u = 1$, we observe that there is a small difference between strategies. Again, the optimal yields the highest error, but the difference is small. When jamming is available, as we noted in the previous section, JFA actually decreases the error committed by EWSZOT, because of the conservative decision when a sensor is jammed.

Again, the ASs can significantly harm the system when $u = 0$. The attack efficiency is significantly enhanced by having the possibility of jamming good sensors (the effects can be dramatic, as in (c) case in Figure 4). We can observe in this case both the problem of ad-hoc defense and the problem of optimality: since AFA is not the optimal attack against EWSZOT, an intelligent choice of when to give false reports and when to give true ones provides an attack that degrades the performance of EWSZOT. And by adding jamming capabilities, the attackers are able to achieve an even higher degradation on the defense mechanism. Yet the main drawback of the optimal strategies is that they are very costly to compute as the CSS network grows. Observe that we have limited to a case with a low dimensionality for our examples in order to avoid this problem.

7. Using the MDP model to learn attack strategies

Now, we turn to explore the possibilities of learning attack strategies using our MDP model. We make use of three different implementations of Q-learning: Q-learning, DQN and DRQN.

Algorithm 6 Q-learning algorithm for the EWSZOT problem

Input: S,A, P_c, M_m, M_j, q, g, M_a, M_g, N, α_0, α_d, ϵ_0, ϵ_d, n_{epochs}

Initialize $Q(s^n, a) = 0, \forall a \in A, \forall s^n \in S$
Set $\alpha = \alpha_0$
Set $\epsilon = \epsilon_0$
for n_{epochs} repetitions **do**
  Initialize $s^n = s^0$
  for $n = 0 : N - 1$ **do**
    Obtain action a using ϵ-greedy policy
    Take action a and obtain s^{n+1} and $r = p_e/N$
    Update $Q(s^n, a)$ using (5)
    Set s^{n+1} as the new state
  Update $\epsilon = \epsilon \cdot \epsilon_d$
  Update $\alpha = \alpha \cdot \alpha_d$
for $s^n \in S$ **do**
  $\hat{\pi}(s^n)^* = \arg\max_a Q(s^n, a)$

Output: $\hat{\pi}(s^n)^*$

7.1. RL using Q-learning

To implement QL, we need to simulate the environment with which our agent will interact: we do so by using Algorithm 2. Recall that we define the state as the tuple $s^n = \langle r_g^n, r_a^n, m_j^n \rangle$ and the action as the tuple $a = \langle a_g, a_a \rangle$. The environment returns the

next state s^{n+1} and the immediate reward obtained r, which is the total error probability p_e: it will be either $p_e = 1/N$ if EWSZOT decided wrongly or $p_e = 0$ if there was no error deciding.

We use variables α and ϵ: we initialize each of these values to $\alpha_0 = 0.5$ and $\epsilon_0 = 1$ respectively and then, after each iteration of the algorithm, we update then by using a decay factor $\alpha_d = \epsilon_d = 0.9997$ (i.e., $\alpha = \alpha \cdot \alpha_d$ and $\epsilon = \epsilon \cdot \epsilon_d$). This decay is used to met the convergence conditions and to balance the exploration-exploitation trade-off: the agent starts with a high ϵ and thus, it explores often. As the agent interacts with the environment, the agent explores less and exploits more: eventually, we want to end with an ϵ value close to 0. This causes that the $Q(s^n, a)$ values are close to the real ones when using a greedy policy (i.e., ϵ-greedy policy when $\epsilon = 0$). In other words, after approximating the Q function, we can approximate the optimal (greedy) policy π^* as:

$$\pi(s^n)^* \approx \hat{\pi}(s^n)^* = \arg\max_a Q(s^n, a) \tag{16}$$

We repeat the learning procedure $n_{epochs} = 2 \cdot 10^4$ times. The whole procedure is summarized in Algorithm 6.

7.2. RL using DQN and DRQN

For DQN, we use a three layers neural network whose structure can be seen in Figure 5. Each of the three layers is fully connected to its neighbors. The first layer has as input size the state size, 24 neurons and uses rectifier linear units for activation (that is, they follow the function $f(x) = \max(0, x)$). The second layer has also 24 neurons and also uses rectifier linear units for activation. The final layer has 24 neurons, an output size equal to the possible number of actions and it uses linear units for activation (i.e., $f(x) = x$).

For DRQN, we use a two layers neural network whose structure can be seen in Figure 5. The first layer is a LSTM (Hochreiter and Schmidhuber, 1997), whose output space has a dimensionality of 32, which takes sequences of 4 time steps. The second layer has 24 neurons, an output size equal to the possible number of actions and it uses linear units for activation.

For both DQN and DRQN, we choose adam as optimizer (Kingma and Ba, 2014), with parameters $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. The loss function we use is the mean squared error (MSE). The maximum number of experiences stored (i.e., the dimension of E) is of 10^4 experiences. After a new experience e^n has been obtained and added to the set E, we randomly pick a mini-batch of 128 experience elements and train the neural network. We use an ϵ-greedy policy with variable ϵ: $\epsilon_0 = 1$ and $\epsilon_d = 0.9995$, with a minimum value $\epsilon = 0.01$. We train the networks using $n_{epochs} = 2 \cdot 10^3$ episodes.

7.3. Results of RL strategies

We simulate using the same CSS network as in the previous simulation ($M = 10$, $M_a = 1$, $M_g = 9$, $M_m = 4$, $q = 2$ and $g = 5.51$). Again, we test for $M_j = \{0, 1, 2\}$, using 51 equispaced values of P_c in the range $P_c \in [0, 0.5]$, using $N = 5$ stages and using $u = 0$ and $u = 1$. We compare the empirical curves obtained averaging 100 runs of the

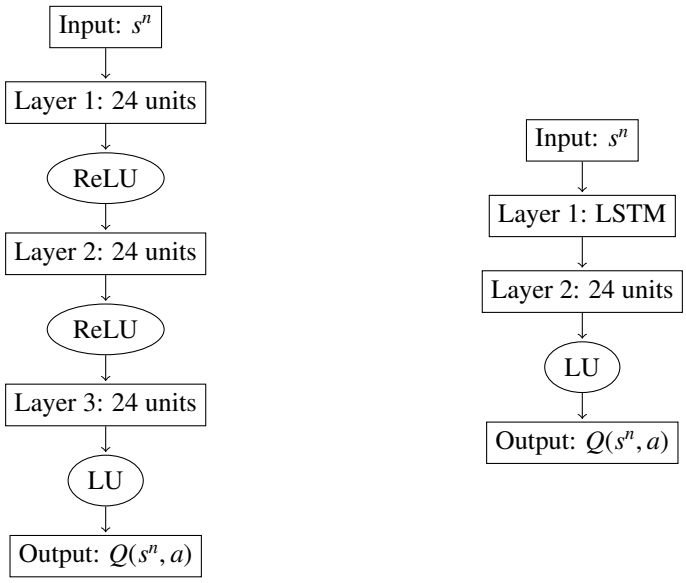


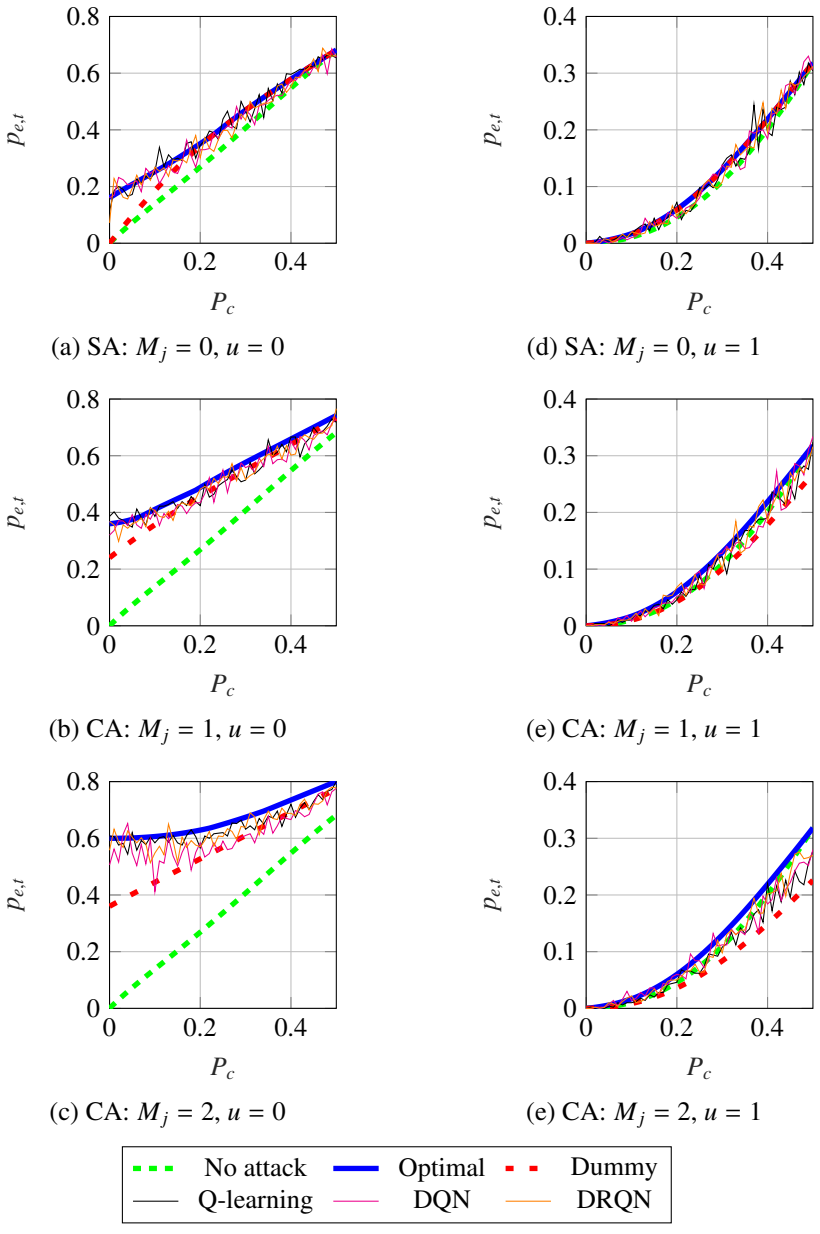
Figure 5: DQN and DRQN structures chosen. For DQN (left), the three layers are fully connected and each of them has 24 units. ReLU is the rectified nonlinearity activation function $f(x) = \max(0, x)$ and LU is the linear activation function $f(x) = x$. For DRQN (right), the first layer is a LSTM with an output space dimensionality of 32, and the second is a dense layer. The input is the state s^n and the output is the estimation of $Q(s^n, a)$.

trained policies obtained using Q-learning, DQN and DRQN. We compare these values with the theoretical curves obtained for the optimal and dummy attacks obtained in the previous simulation. The results can be observed in Figure 6. Note that Q-learning, DQN and DRQN provide very good results, with error curves similar to the optimal theoretical ones, with a low variance, for all cases. Being the results quite similar, DQN and DRQN presents an advantage: they took less computation time to learn than Q-learning, around one magnitude order below (and DQN learned slightly faster than DRQN). This is mainly due to the use of experience replay: Q-learning trained using 10 times more epochs than DQN and DRQN. Another advantage of DQN and DRQN is that they do not need memory to store the Q-function, although they require memory to store the E set. Note , however, that E has a size limited that we choose before training, but Q-value function entries number grows exponentially with the problem parameters (see Section 4.5). Hence, DQN and DRQN are a more versatile solution for high dimensionality problems, where Q-learning may not be practical due to the memory cost.

8. Comparison of attack strategies

We compare the strategies used using the following points, that are summarized in Table 1:

- In terms of complexity to obtain the policy, dummy strategies are simple since



(a) SA: $M_j = 0$, $u = 0$

(d) SA: $M_j = 0$, $u = 1$

(b) CA: $M_j = 1$, $u = 0$

(e) CA: $M_j = 1$, $u = 1$

(c) CA: $M_j = 2$, $u = 0$

(e) CA: $M_j = 2$, $u = 1$

Figure 6: Performance of optimal, simple and learned strategies against EWSZOT in terms of $p_{e,t}$ as a function of P_c. We compare the results of the three RL algorithms with the optimal and dummy strategies theoretical values. Observe that all RL algorithms learn strategy that is quasi-optimal.

	Dummy	Optimal	Q-learning	DQN/DRQN
Policy obtaining	Easy	Very hard	Hard	Medium
Computation	Low	Very high	High	Medium
Implementation	Very easy	Easy	Easy	Medium
Attack results	Poor	Optimal	Very good	Very good

Table 1: Comparison of different strategies against EWSZOT.

they are fixed beforehand. The optimal approach on the other hand is hard: we must know both the state space and the transitions, and that is costly. Q-learning requires knowing all possible states: these might be hard to compute. Finally, DQN and DRQN do not require knowing neither the states nor the transitions, and thus, the difficulty in this case is lower than using Q-learning; yet tuning the hyperparameters for training might be tricky for this approach.

- In terms of computational resources, dummy strategies have very small requirements both in computation and memory capacities. Optimal strategies need a large amount of both. Q-learning is not as computationally expensive but still needs a huge amount of memory. DQN and DRQN do not require as much computational capacity as the optimal strategies, and their memory requirements are lower and indeed, could be controlled by adjusting the size of E (although this would influence the training process).

- In terms of policy implementation, dummy strategies are very easy to implement due to being fixed. The optimal and Q-learning strategies can be implemented as a search over a table that returns the prescribed action for a given state. DQN and DRQN require implementing a neural network, which causes it to be a bit harder.

- In terms of attack results (i.e., the total error probability), dummy strategies give the worst results. The best possible result, by definition, correspond to the optimal strategies. And Q-learning, DQN and DRQN provide quasi-optimal solutions, similar to the optimal ones.

By comparing all strategies, we see that dummy and optimal strategies do pose a strong trade-off between attack results and complexity. But this trade-off can be alleviated by using RL strategies, specially DQN and DRQN. Using neural networks provides a

good compromise between complexity and attack results: the algorithm is not too hard to implement, it is not excessively expensive computationally and finally, its results are quasi-optimal.

Note that our results show that current WSN defense mechanisms can be vulnerable to an intelligent attacker, specially to one based on Deep RL tools. As we just showed, such an attacker can obtain good attack results - i.e., can degrade the defense mechanism performance - even if it does not know the concrete defense mechanism used. The computational cost for that attacker may be under the computational capacities of current hardware. We believe, hence, that further research is needed in order to obtain defense mechanisms that are able to tackle with these intelligent attackers.

9. Conclusions

In this paper, we have proposed using MDPs as a tool to model and study attacks in WSN. We have shown that current defense mechanism present two related problems, namely, the problem of ad-hoc defense and the problem of optimality. An attacker making use of our MDP approach could successfully exploit these problems and degrade a defense mechanism, as our study of EWSZOT shows. The main strengths of our approach are:

- It is possible to obtain the theoretical results of different attack policies if the MDP is modeled, and even obtaining the optimal attack policies. Note that this means that different attacks can be compared in terms of their effects in a straightforward way. It also addresses the optimality problem: we can obtain the optimal attack and hence, have a bound on the defense mechanism performance.

- The use of RL tools allows us to obtain quasi-optimal attack results if the MDP cannot be modeled due to being unknown or if it is highly complex. We have shown that Deep RL tools are of special interest here due to the balance between results achieved and their complexity. We remark that RL tools exploit the problem of ad-hoc defense: minor changes in the attack mechanism (such as giving false reports selectively in case of EWSZOT) may cause a significant degradation in the defense mechanism.

Our approach also has some weaknesses:

- Modeling an MDP may be very hard: the state-action space could be prohibitively large or the transition function could be very complex to obtain. Hence, optimal solutions could be computationally very complex to obtain. Also, note that minimal changes in the MDP may significantly change the MDP definition and transition function. Note that this is a strength for RL methods: they can be adapted easily to such changes.

- Q-learning suffers also when the action-space is large.

- In our work, we have focused in the single agent case (i.e., only one AS), and we have also focused on discrete actions. Thus, our results only apply when these conditions hold.

- We have also assumed that the ASs can observe the state of the system (the reputations in case of EWSZOT). This assumption need not hold in real-life systems, where only a noisy observation of the state could be available.

- Deep RL methods results can be sensitive to hyperparameter tuning and also, to the reward scheme. That is, different reward schemes may cause the agent to learn different attacks. In our example, the reward definition was straightforward - i.e., the total error probability -, but we assumed that the ASs knew the channel state, which need not happen in real-life systems.

Finally, we believe that our framework could be extended in order to address some of these weaknesses. We identify the following possible future work lines:

- It could be possible to extend our work to work without an explicit knowledge of the state. In this case, it would be needed to solve a Partially Observable MDP (POMDP), which are significantly harder to solve analytically (Thrun et al., 2005). However, Deep RL tools could be very helpful to manage POMDPs, specially DRQN, as shown in (Hausknecht and Stone, 2015).

- Another possible extension could be addressed to manage a continuous action space, specially when an approximation based on discretization would significantly enlarge the action space. Note that in many attacks, it could make more sense using continuous actions, for instance, to model probabilities. In this sense, there are Deep RL algorithms other than DQN and DRQN that can cope with continuous action spaces, such as Trust Region Policy Optimization (Schulman et al., 2015).

- It would be interesting also extending the work towards having more than one ASs. It would be of special interest exploring the effects that communication among ASs may have on the defense mechanism degradation, as well as the possible improvements with regards to having a single AS.

- Finally, we think that the most important extension has to do with defense mechanisms that are able to cope with intelligent attackers. In this case, it would be necessary to use tools from dynamic game theory (Littman, 1994), in which we have several agents, each one solving an MDP that is coupled to the other agents. There are significant challenges related to this setup, as (Hernandez-Leal et al., 2017) shows. However, it is imperative to develop defense mechanisms that are able to cope with such intelligent attackers: as we have shown, many current WSN defense mechanisms suffer from the problems of the ad-hoc defense and optimality, which an intelligent attacker may exploit.

10. Acknowledgements

This work was supported by a Ph.D. grant given to the first author by Universidad Politécnica de Madrid, as well as by the Spanish Ministry of Science and Innovation under the grant TEC2016-76038-C3-1-R (HERAKLES) and the COMONSENS Network of Excellence TEC2015-69648-REDC.

References

Alsheikh, M.A., Lin, S., Niyato, D., Tan, H.P., 2014. Machine learning in wireless sensor networks: Algorithms, strategies, and applications. IEEE Communications Surveys & Tutorials 16, 1996–2018.

Aref, M.A., Jayaweera, S.K., Machuzak, S., 2017. Multi-agent reinforcement learning based cognitive anti-jamming, in: Wireless Communications and Networking Conference (WCNC), 2017 IEEE, IEEE. pp. 1–6.

Benedetto, F., Tedeschi, A., Giunta, G., Coronas, P., 2016. Performance improvements of reputation-based cooperative spectrum sensing, in: Personal, Indoor, and Mobile Radio Communications (PIMRC), 2016 IEEE 27th Annual International Symposium on, IEEE. pp. 1–6.

Bertsekas, D.P., 1995. Dynamic programming and optimal control. volume 1. Athena Scientific Belmont, MA.

Cichoń, K., Kliks, A., Bogucka, H., 2016. Energy-efficient cooperative spectrum sensing: A survey. IEEE Communications Surveys & Tutorials 18, 1861–1886.

Fragkiadakis, A.G., Tragos, E.Z., Askoxylakis, I.G., 2013. A survey on security threats and detection techniques in cognitive radio networks. IEEE Communications Surveys & Tutorials 15, 428–445.

Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y., 2016. Deep learning. volume 1. MIT press Cambridge.

Goumagias, N.D., Hristu-Varsakelis, D., Assael, Y.M., 2018. Using deep q-learning to understand the tax evasion behavior of risk-averse firms. Expert Systems with Applications 101, 258–270.

Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R., Schmidhuber, J., 2017. Lstm: A search space odyssey. IEEE transactions on neural networks and learning systems 28, 2222–2232.

Han, G., Xiao, L., Poor, H.V., 2017. Two-dimensional anti-jamming communication based on deep reinforcement learning, in: Proceedings of the 42nd IEEE International Conference on Acoustics, Speech and Signal Processing,.

Hausknecht, M., Stone, P., 2015. Deep recurrent q-learning for partially observable mdps. Proc. of Conf. on Artificial Intelligence, AAAI, 2015. .

Hernandez-Leal, P., Kaisers, M., Baarslag, T., de Cote, E.M., 2017. A survey of learning in multiagent environments: Dealing with non-stationarity. arXiv preprint arXiv:1707.09183 .

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural computation 9, 1735–1780.

Jeong, G., Kim, H.Y., 2018. Improving financial trading decisions using deep q-learning: Predicting the number of shares, action strategies, and transfer learning. Expert Systems with Applications .

Kailkhura, B., Brahma, S., Varshney, P.K., 2014. On the performance analysis of data fusion schemes with byzantines, in: Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on, IEEE. pp. 7411–7415.

Kingma, D., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 .

Li, J., Feng, Z., Wei, Z., Feng, Z., Zhang, P., 2014. Security management based on trust determination in cognitive radio networks. EURASIP Journal on Advances in Signal Processing 2014, 48.

Li, Y., Quevedo, D.E., Dey, S., Shi, L., 2017. Sinr-based dos attack on remote state estimation: A game-theoretic approach. IEEE Transactions on Control of Network Systems 4, 632–642.

Littman, M.L., 1994. Markov games as a framework for multi-agent reinforcement learning, in: Machine Learning Proceedings 1994. Elsevier, pp. 157–163.

Liu, S., Liu, Q., Gao, J., Guan, J., 2011. Attacker-exclusion scheme for cooperative spectrum sensing against ssdf attacks based on accumulated suspicious level, in: Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2011 IEEE International Conference on, IEEE. pp. 239–243.

Min, A.W., Shin, K.G., Hu, X., 2009. Attack-tolerant distributed sensing for dynamic spectrum access networks, in: Network Protocols, 2009. ICNP 2009. 17th IEEE International Conference on, IEEE. pp. 294–303.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 .

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al., 2015. Human-level control through deep reinforcement learning. Nature 518, 529–533.

Mpitziopoulos, A., Gavalas, D., Konstantopoulos, C., Pantziou, G., 2009. A survey on jamming attacks and countermeasures in wsns. IEEE Communications Surveys & Tutorials 11.

Ndiaye, M., Hancke, G.P., Abu-Mahfouz, A.M., 2017. Software defined networking for improved wireless sensor network management: A survey. Sensors 17, 1031.

Nguyen-Thanh, N., Koo, I., 2009. An enhanced cooperative spectrum sensing scheme based on evidence theory and reliability source evaluation in cognitive radio context. IEEE Communications Letters 13.

Noon, E., Li, H., 2010. Defending against hit-and-run attackers in collaborative spectrum sensing of cognitive radio networks: A point system, in: Vehicular Technology Conference (VTC 2010-Spring), 2010 IEEE 71st, IEEE. pp. 1–5.

Rawat, P., Singh, K.D., Chaouchi, H., Bonnin, J.M., 2014. Wireless sensor networks: a survey on recent developments and potential synergies. The Journal of supercomputing 68, 1–48.

Sampath, A., Dai, H., Zheng, H., Zhao, B.Y., 2007. Multi-channel jamming attacks using cognitive radios, in: Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on, IEEE. pp. 352–357.

Sarigiannidis, P., Karapistoli, E., Economides, A.A., 2015. Detecting sybil attacks in wireless sensor networks using uwb ranging-based information. Expert Systems with Applications 42, 7560–7572.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P., 2015. Trust region policy optimization, in: International Conference on Machine Learning, pp. 1889–1897.

Sokullu, R., Dagdeviren, O., Korkmaz, I., 2008. On the ieee 802.15. 4 mac layer attacks: Gts attack, in: Sensor Technologies and Applications, 2008. SENSORCOMM'08. Second International Conference on, IEEE. pp. 673–678.

Sutskever, I., 2013. Training recurrent neural networks. University of Toronto, Toronto, Ont., Canada .

Sutton, R.S., Barto, A.G., 1998. Reinforcement learning: An introduction. volume 1. MIT press Cambridge.

Thrun, S., Burgard, W., Fox, D., 2005. Probabilistic robotics. MIT press.

Tomić, I., McCann, J.A., 2017. A survey of potential security issues in existing wireless sensor network protocols. IEEE Internet of Things Journal 4, 1910–1923.

Wang, S.S., Yan, K.Q., Wang, S.C., Liu, C.W., 2011. An integrated intrusion detection system for cluster-based wireless sensor networks. Expert Systems with Applications 38, 15234–15243.

Wang, W., Sun, Y., Li, H., Han, Z., 2010. Cross-layer attack and defense in cognitive radio networks. Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE , 1–6.

Xiao, L., Li, Y., Huang, X., Du, X., 2017. Cloud-based malware detection game for mobile devices with offloading. IEEE Transactions on Mobile Computing 16, 2742–2750.

Xiao, L., Li, Y., Liu, G., Li, Q., Zhuang, W., 2015. Spoofing detection with reinforcement learning in wireless networks, in: Global Communications Conference (GLOBECOM), 2015 IEEE, IEEE. pp. 1–5.

Xiao, L., Xie, C., Chen, T., Dai, H., Poor, H.V., 2016. A mobile offloading game against smart attacks. IEEE Access 4, 2281–2291.

Yan, Q., Li, M., Jiang, T., Lou, W., Hou, Y.T., 2012. Vulnerability and protection for distributed consensus-based spectrum sensing in cognitive radio networks, in: INFOCOM, 2012 Proceedings IEEE, IEEE. pp. 900–908.

Yang, K., 2014. Wireless sensor networks. Springer.

Yu, F.R., Tang, H., Huang, M., Li, Z., Mason, P.C., 2009. Defense against spectrum sensing data falsification attacks in mobile ad hoc networks with cognitive radios, in: Military Communications Conference, 2009. MILCOM 2009. IEEE, IEEE. pp. 1–7.

Zhang, L., Ding, G., Wu, Q., Zou, Y., Han, Z., Wang, J., 2015. Byzantine attack and defense in cognitive radio networks: A survey. IEEE Communications Surveys & Tutorials 17, 1342–1363.

Zhu, F., Seo, S.W., 2009. Enhanced robust cooperative spectrum sensing in cognitive radio. Journal of Communications and Networks 11, 122–133.