Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Towards probabilistic intrusion detection in supervisory control of discrete event systems [1]

Rômulo Meira-Góes * Christoforos Keroglou ** Stéphane Lafortune *

* *Department of Electrical Engineering and Computer Science*
University of Michigan, Ann Arbor, USA
e-mail: {romulo,stephane}@umich.edu
** *Division of Decision and Control Systems, School of Electrical*
Engineering and Computer Science
KTH Royal Institute of Technology, Stockholm, Sweden
e-mail: keroglou@kth.se

Abstract: In control systems, sensor deception is a class of attacks where an attacker manipulates sensor readings to cause damage to the system. Our work investigates quantitative measurements to detect this class of attacks in the context of stochastic supervisory control. We introduce the notion of ϵ-safe systems, which is a first step to generalize qualitative intrusion detection conditions to quantitative intrusion detection conditions. We provide sufficient and necessary conditions to verify if a system is ϵ-safe. Moreover, we provide an algorithm that verifies these conditions, which implies that the problem is decidable.

Keywords: Supervisory control and automata; Discrete event modeling and simulation; Intrusion detection; Security;

1. INTRODUCTION

The control community recently started to incorporate security aspects into the design of feedback control systems (Cardenas et al., 2008; Teixeira et al., 2012). Understanding and designing feedback control systems that are robust against attacks is of critical importance nowadays. A key step towards feedback control systems that are robust against attacks is an intrusion detection system. This paper focuses on the design of better intrusion detection modules for control systems. We investigate this problem in the context of stochastic supervisory control theory, where the underlying uncontrolled system has been abstracted as a stochastic discrete transition system (the *plant* in this work), where sensor outputs belong to a finite set of events. Driven by these events, a high-level supervisory controller, or simply *supervisor*, controls the behavior of the plant via actuator commands.

Based on this event-driven model, we incorporate an attacker that hijacks a subset of the events and sends to the supervisor incorrect information about the plant's sensors; this type of attack is known as *sensor deception attack*. In this scenario, an intrusion detection module monitors the behavior of the controlled system and decides if an attacker is disrupting the nominal controlled behavior.

Prior work on security against sensor deception attacks in the field of Discrete Event Systems (DES) (Rashidinejad et al., 2019) mainly focuses on designing attack strategies for *fixed supervisors* (Meira-Góes et al., 2017; Su, 2018; Meira-Góes et al., 2019a,b), on designing intrusion detection modules for *fixed supervisors* (Thorsley and Teneketzis, 2006; Carvalho et al., 2018; Lima et al., 2019) or designing robust supervisors (Su, 2018; Meira-Góes et al., 2019c; Wang et al., 2020). Some of these works considered stochastic models (Thorsley and Teneketzis, 2006; Meira-Góes et al., 2019b), while the remainder considered logical models.

In (Thorsley and Teneketzis, 2006), the attacker rewrites actuator commands with given probabilities. Although these probabilities generate a probabilistic measure on the controlled system, the intrusion detection problem is investigated under a logical framework. The works of (Carvalho et al., 2018; Lima et al., 2019) extend the intrusion detection problem for general attack models, including sensor and actuator attacks, under the same logical framework.

Since intrusion detection in the context of logical models is a strong requirement, there is a need to develop quantitative frameworks for detection of sensor deception attacks, to complement the logical approach. To investigate this problem, we propose to adopt a stochastic framework where the plant is a stochastic automaton under the control of a logical deterministic supervisor. In this context, we are able to calculate likelihoods of attacks based on strings observed by the supervisor.

The stochastic framework adopted is similar to the one in (Meira-Góes et al., 2019b) and is inspired by the prior work in (Kumar and Garg, 2001). We introduce the notion of ϵ-safe systems which is a first step to generalize the qualitative notions of logical intrusion detection to quantitative notions of intrusion detection. The definition of ϵ-safety captures quantitatively the attacks that are undetectable by a logical intrusion detection module that operates on the controlled system, assuming a fixed supervisor and a set of compromised events (sensors). Intuitively, an attacker might leave a detectable probabilistic trace when it modifies the nominal controlled behavior.

The paper is organized as follows. Section 2 introduces the necessary background used throughout the paper. The framework of supervisory control theory under sensor deception attacks is presented in Section 3. Section 4 presents the definition of ϵ-safe systems and the verification problem of this property. In

[1] The work of R.M.G. and S.L. is supported by US NSF grant CNS-1738103.

Section 5, we provide the solution methodology for the studied problem and discuss its correctness. We conclude the paper in Section 6.

2. PRELIMINARIES

2.1 Supervisory control

We consider the supervisory level of a feedback control system, where the uncontrolled system is modeled as a Deterministic Finite-State Automaton (DFA) in the discrete-event modeling formalism. A DFA is denoted by $G = (X_G, \Sigma, \delta_G, x_{0,G}, X_{G,m})$, where X_G is the finite set of states, Σ is the finite set of events, $\delta_G : X_G \times \Sigma \to X_G$ is the partial transition function, $x_{0,G}$ is the initial state and $X_{G,m}$ is the set of marked states. The function δ_G is extended, in the usual manner, to the domain $X_G \times \Sigma^*$.

For any string $s \in \Sigma^*$, $s[i]$ denotes the i^{th} event of s such that $s = s[1]s[2] \dots s[|s|]$, where $|s|$ denotes the length of s. The i^{th} prefix of s is denoted by s^i, namely $s^i = s[1] \dots s[i]$ and $s^0 = \epsilon$. Finally, $\mathbb{N}$ is the set of natural numbers, $[n]$ is the set of natural numbers bounded by n and $[n]^+$ is the set of positive natural numbers bounded by n.

The language and the marked language generated by G are defined by $\mathcal{L}(G) = \{s \in \Sigma^* | \delta_G(x_{0,G}, s)!\}$ and $\mathcal{L}_m(G) = \{s \in \mathcal{L}(G) | \delta_G(x_{0,G}, s) \in X_{G,m}\}$, where ! means that the function is defined for these arguments. A string $s \in \mathcal{L}(G)$ generates a unique run $x_1 s[1] x_2 \dots s[n] x_{|s|+1}$, where $x_{i+1} = \delta_G(x_i, s[i])$ for $i \in [|s|]^+$ and $x_1 = x_{0,G}$. A sub-run in G is defined as $x_1 s[1] x_2 \dots s[n] x_{|s|+1}$, where $x_{i+1} = \delta_G(x_i, s[i])$ for $i \in [|s|]^+$ and $x_1 \in X_G$. The active event set of state $x \in X_G$ is defined as $\Gamma_G(x) = \{\sigma \in \Sigma | \delta_G(x, \sigma)!\}$. Lastly, the operation $CoAc(G) = (X_{CoAc(G)}, \Sigma, \delta_{CoAc(G)}, x_{0,CoAc(G)}, X_{G,m})$, as in (Cassandras and Lafortune, 2008), returns the coaccessible part of automaton G, i.e., $\mathcal{L}(G) = pre(\mathcal{L}_m(G))$, where $pre(L)$ returns all the prefixes of language L.

The system G, in supervisory control theory, is considered as the *plant* (uncontrolled system) that needs to be controlled to meet a desired specification. The limited control capabilities in G are characterized by partitioning the set Σ into two disjoint sets, the set of controllable events Σ_c and the set of uncontrollable events Σ_{uc}.

A supervisor dynamically enables/disables controllable events of the plant such that it generates a controlled behavior that satisfies a desired specification. A supervisor is formally defined as a function $S : \Sigma^* \to \Gamma$, where $\Gamma = \{\gamma \subseteq \Sigma | \Sigma_{uc} \subseteq \gamma\}$ is the set of admissible control decisions. The closed-loop behavior of the controlled system is denoted by S/G and defined by the language $\mathcal{L}(S/G)$; see, e.g., (Cassandras and Lafortune, 2008). Without loss of generality, S is realized by an automaton $R = (X_R, \Sigma, \delta_R, x_{0,R})$.

2.2 Stochastic supervisory control

We consider a stochastic DES modeled as a Probabilistic Finite-State Automaton (PFA). A PFA is denoted by $H = (X_H, \Sigma, P_H, x_{0,H}, X_{H,m})$, where X_H, Σ, $x_{0,H}$ and $X_{H,m}$ are defined as in a DFA, $P_H : X_H \times \Sigma \times X_H \to [0,1]$ is the transition probability function. The probability function $P_H(x, \sigma, y)$ specifies the probability of moving from state $x \in X_H$ to state $y \in X_H$ with event $\sigma \in \Sigma$. When it is convenient, we use the notation $P_H^{x,\sigma,y} = P_H(x, \sigma, y)$ and we write $P_H^\sigma(x)$ when $\exists y \in X_H$ such that $P_H^{x,\sigma,y} > 0$. We only consider the case of nonterminating PFA, i.e., for any $x \in X_H$ we have that $\sum_{\sigma \in \Sigma} P_H^\sigma(x) = 1$. This assumption is without loss of generality, as any terminating PFA can be transformed into a nonterminating one, as shown in (Lawford and Wonham, 1993).

The function δ_H is defined to bridge the gap between a PFA and a DFA, where $\delta_H(x, \sigma) = y$ if $P_H^{x,\sigma,y} > 0$. In this work, we assume that δ_H is deterministic, i.e., there does not exist $y, y^* \in X_H$, $y^* \neq y$, such that $P_H^{x,\sigma,y} > 0$ and $P_H^{x,\sigma,y^*} > 0$. Using this definition, every PFA H is associated to a corresponding DFA G where $\delta_G(x, \sigma) = \delta_H(x, \sigma)$ and $\mathcal{L}(H) := \mathcal{L}(G)$. For simplicity, whenever we use a DFA operator in a PFA H, it means that we are analyzing the corresponding DFA G.

Finally, the notion of probabilistic languages (p-languages) of a PFA was introduced in (Garg et al., 1999). Formally, $L_p(H) : \Sigma^* \to [0,1]$ is defined for $s \in \Sigma^*$ and $\sigma \in \Sigma$ as :

$$L_p(H)(\epsilon) = 1 \tag{1}$$

$$L_p(H)(s\sigma) = \begin{cases} L_p(H)(s) P_H^{x,\sigma,y} & \text{if } x = \delta_H(x_{0,H}, s) \\ & \quad y = \delta_H(x_{0,H}, s\sigma) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

In stochastic supervisory control theory, the system H is considered as the plant but there are different manners of studying its closed-loop behavior (Lawford and Wonham, 1993; Kumar and Garg, 2001; Pantelic et al., 2014). In this paper, we use the results of supervisory control of stochastic DES introduced by (Kumar and Garg, 2001), where only the plant behaves stochastically. Namely, both the specification and the supervisor are deterministic and defined as in the previously-described supervisory control framework. However, the supervisor *alters* the probabilistic behavior of the plant via the control actions it takes (disabling events). Conditions for the existence of a supervisor for the above control problem are provided in (Kumar and Garg, 2001).

Formalizing the previous discussion, the disablement of events by R increases the probability of the enabled ones. In other words, R/H generates another p-language, in general, different than the p-language of H. Given a state $x \in X_H$, a state $y \in X_R$, and an event $\sigma \in \Gamma_H(x) \cap \Gamma_R(y)$, the probability of σ being executed is given by the standard normalization:

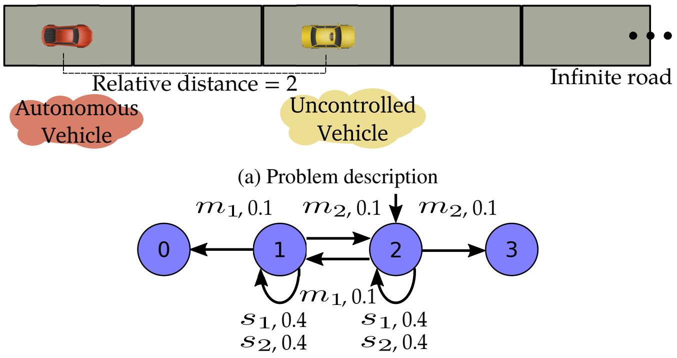
$$P_{x,y}^\sigma = \frac{P_H^\sigma(x)}{\sum_{\sigma' \in \Gamma_H(x) \cap \Gamma_R(y)} P_H^{\sigma'}(x)} \tag{3}$$

We define $M_n = H ||_p R$ as the PFA that describes the behavior of R/H, where $||_p$ is defined based on Equation (3) and the standard parallel composition $||$ (see (Cassandras and Lafortune, 2008)). Formally, $M_n = (X_{M_n}, \Sigma, P_{M_n}, x_{0,M_n})$ is defined by $X_{M_n} \subseteq X_H \times X_R$, $x_{0,M_n} = (x_{0,H}, x_{0,R})$, and for $x = (x_1, x_2)$, $y = (y_1, y_2) \in X_H \times X_R$ and $\sigma \in \Sigma$ the transition probability is:

$$P_{M_n}(x, \sigma, y) = \begin{cases} P_{x_1,x_2}^\sigma & \text{if } \delta_H(x_1, \sigma) = y_1 \wedge \\ & \quad \delta_R(x_2, \sigma) = y_2 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

For simplicity and without loss of generality, we assume that the plant H has one critical state, denoted $x_{crit} \in X_H$ and supervisor R ensures that this state is not reachable in R/H. We define the set of unsafe strings as $L_{crit} = \{s \in \Sigma^* \mid P_H(x_{0,H}, s, x_{crit}) > 0\}$.

Example 1. We assume that two vehicles are traveling in the same direction on an infinite road as shown in Fig. 1(a). The vehicle in front is assumed to be manually driven while the vehicle behind is assumed to be autonomous. Instead of their exact position on the road, we represent their state by their relative position, i.e., the difference of their exact positions. The vehicles could either *stay* in their positions, events s_1 and s_2, or *move* to an adjacent position, events m_1 and m_2. For simplicity, we assume that once the relative distance is greater than or equal to three, or equal to zero the experiment ends. Figure 1(b) models this problem as a PFA, where the events of the autonomous vehicle are controllable, $\Sigma_c = \{s_1, m_1\}$, the probability transition function is encoded in the transition arcs and the name of the states is the relative distance between the vehicles.



(a) Problem description

(b) Model of the autonomous vehicle problem

Fig. 1. Simplified autonomous vehicle example

The goal of the autonomous vehicle is to avoid crashing into the front car, i.e., state 0 is critical. The corresponding DFA of the model in Fig. 1(b) without state 0 provides the supremal supervisor for this problem. The closed-loop controlled behavior M_n is shown in Fig. 2.

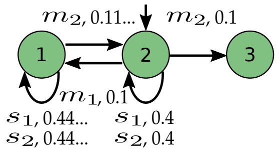


Fig. 2. The controlled system M_n

3. ATTACKED SYSTEM DESCRIPTION

3.1 Notation

We define $\Sigma_a \subseteq \Sigma$ as the set of compromised events, i.e., the attacker can insert/delete events in this set on the communication channel. We use subscripts to identify attacker modifications; the sets $\Sigma_i = \{e_i \mid e \in \Sigma_a\}$ and $\Sigma_d = \{e_d \mid e \in \Sigma_a\}$ are the sets of inserted and deleted events, respectively. Events without subscripts are legitimate events generated by the plant H, whereas events with subscripts are events altered by the attacker. For convenience, let $\Sigma_e = \Sigma_i \cup \Sigma_d$ be the editable event set and $\Sigma_m = \Sigma \cup \Sigma_e$ the complete event set.

The mask $\mathcal{M} : \Sigma_m \to \Sigma$ removes the subscripts from events in Σ_e, i.e., $\mathcal{M}(e_d) = \mathcal{M}(e_i) = e$. Let Π^H (Π^R) be a projection operator that projects events in Σ_m to events in Σ generated by the plant (observed by the supervisor). Namely, Π^H outputs the event that is executed in H, i.e., $\Pi^H(e_i) = \epsilon$ and $\Pi^H(e_d) = \Pi^H(e) = e$. On the other hand, Π^R outputs the event observed by the supervisor, i.e, $\Pi^R(e_d) = \epsilon$ and $\Pi^R(e_i) = \Pi^R(e) = e$. Lastly, strings $s_H = \Pi^H(s)$ and $s_R = \Pi^R(s)$ are the plant projection and the supervisor projection of string $s \in \Sigma_m^*$.

3.2 Attacked controlled systems

In this work, sensor deception attacks are considered, where an attacker hijacks a subset of the sensors and modifies them in order to reach a specific goal. Similarly as in (Meira-Góes et al., 2019b), we assume that an attacker is modeled as a deterministic attack function $f_A : \Sigma_m \to \Sigma_m^*$ with constraints based on the set of compromised events Σ_a (Meira-Góes et al., 2019b). For simplicity and without loss of generality, we assume that an attack function is given as an automaton $A = (X_A, \Sigma_m, \delta_A, x_{0,A})$; for more details on the definition of the attack function see (Meira-Góes et al., 2019a,b). Intuitively, the attacker observes events from the plant and remembers its previous modifications to decide the modified string it will send to the supervisor.

The presence of a general sensor deception attacker defined by A disturbs the *nominal* behavior of controlled system. In fact, a new controlled system, denoted as *attacked system*, is produced in the presence of an attacker. We review the definition of this attacked system. First, we need to modify automata H and R so that they include attack actions.

Definition 1. Given H and Σ_a, we define the attacked plant H_a as: $H_a = (X_{H_a}, \Sigma_m, P_{H_a}, x_{0,H_a})$

1: $X_{H_a} = X_H$
2: Σ_m
3: $$P_{H_a}^{(x,e,y)} = \begin{cases} P_H^{(x,e,y)} & \text{if } e \in \Sigma \text{ and } \delta_H(x,e) = y \\ 1 & \text{if } e \in \Sigma_i \text{ and } x = y \\ P_H^{(x,\mathcal{M}(e),y)} & \text{if } e \in \Sigma_d \text{ and} \\ & \delta_H(x,\mathcal{M}(e)) = y \\ \text{undefined} & \text{otherwise} \end{cases}$$
where $x, y \in X_{H_a}$ and $e \in \Sigma_m$
4: $x_{0,H_a} = x_{0,H}$

Note that, H_a violates $\sum_{e \in \Sigma_m} P_{H_a}^e(x) = 1$ for any $x \in X_{H_a}$ since we introduce the insertion events with probability one and deletion events with the same probability as their legitimate events. Nonetheless, we do not analyze H_a by itself as it is just an intermediate step.

Similarly to the construction of H_a, we modify the behavior of R to reflect the modifications made by an attacker on the communication channel. We assume that R respects the controllability condition (Cassandras and Lafortune, 2008).

Definition 2. Given R and Σ_a, we define the attacked supervisor $R_a = (X_{R_a}, \Sigma_m, \delta_{R_a}, x_{0,R_a})$ as:

1: $X_{R_a} = X_R$
2: Σ_m
3: $$\delta_{R_a}(x,e) = \begin{cases} \delta_R(x,e) & \text{if } e \in \Sigma \text{ and } \delta_R(x,e)! \\ x & \text{if } e \in \Sigma_d \text{ and } \delta_R(x,\mathcal{M}(e))! \\ \delta_R(x,\mathcal{M}(e)) & \text{if } e \in \Sigma_i \text{ and } \delta_R(x,\mathcal{M}(e))! \\ \text{undefined} & \text{otherwise} \end{cases}$$
where $x \in X_{R_a}$ and $e \in \Sigma_m$
4: $x_{0,R_a} = x_{0,R}$

Based on H_a, R_a and A, we define the attacked system as $M_a = H_a||_p(R_a||A)$. We used the parentheses with $(R_a||A)$ to remind that A and R generate a new supervisor that supervises H_a. The PFA M_a defines the language of the attacked system in Σ_m^*, i.e., with the subscripts for each attacker modification. For convenience, we define:

$$X_{crit,a} = \{x \in X_{M_a} \mid \exists s \in \mathcal{L}(M_a) \text{ s.t. } x_{crit} = \delta_{G_a}(x_{0,G_a}, s)\}$$

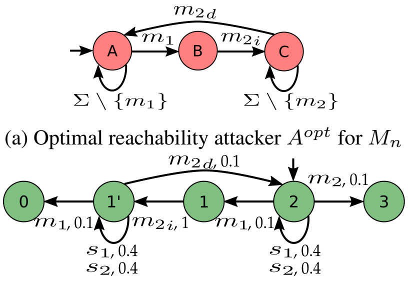
Remark 1: Although it could be that the attacker acts as a supervisor in the composition $H_a||_p(R_a||A)$, we only consider attackers A that respect controllability whenever the attacker does not make an insertion. When an attacker inserts an event, it is assumed that the attacker acts faster than the plant, i.e., the plant is "blocked" to execute events during this short period of "time". On the other hand, the attacker does not disable any plant event when it does not insert an event. It is easy to check if an attacker satisfies these conditions (Meira-Góes et al., 2019b).

Remark 2: The PFA M_a is versatile since two useful languages other than $\mathcal{L}(M_a)$ are easily extracted from it. Namely, the attacked language executed by H is obtained by $\Pi^H(\mathcal{L}(H_a \|_p(R_a\|A)))$ while the language seen by the supervisor is obtained by $\Pi^R(\mathcal{L}(H_a\|_p(R_a\|A)))$.

In (Meira-Góes et al., 2019b), the notion of *winning level of an attacker* A is defined to be the probability that M_a generates unsafe strings. Namely, $win_A = \sum_{s\in\tilde{L}_{crit,a}} L_p(M_a)(s)$ [2] , where $L_{crit,a} = \{s \in \Sigma_m^* | \delta_{M_a}(x_{0,M_a}, s) \in X_{crit,a}\}$. Moreover, an optimal attack function A^{opt}, one with the largest win_A, exists, is realizable, and is deterministic and memoryless. We call an attacker that implements an optimal attack function as an optimal reachability attacker.

In this paper, we focus on investigating the detection of these optimal strategies. In other words, the attacker might leave a probabilistic trace in order to achieve an optimal result. This trace could be used to detect if the controlled system is under attack.

Example 2. Back to our running example, we assume that an attacker manipulates events $\Sigma_a = \{s_2, m_2\}$. Figure 3(a) illustrates an optimal reachability attack strategy, where the attacker simply inserts event m_2 when the relative distance between the vehicles is 1. The attacked system M_a is depicted in Fig. 3(b) and using the results in (Meira-Góes et al., 2019b), we get that $win_{A^{opt}} = 0.5$.



(a) Optimal reachability attacker A^{opt} for M_n

(b) Attacked system M_a

Fig. 3. Optimal attack strategy and attacked system

4. PROBABILISTIC INTRUSION DETECTION PROBLEM

4.1 Intuition on the problem formulation

In (Meira-Góes et al., 2019b), an attacker is detected when the attacked system generates a string with a supervisor projection outside of $\mathcal{L}(M_n)$. In fact, the detection modules defined in (Lima et al., 2019) and (Carvalho et al., 2018) only detect these strings. Nonetheless, strings that reach the critical state and whose Π^R projection belongs to $\mathcal{L}(M_n)$ are undetectable by these detection modules. Our objective is to use the probabilistic information about these strings, that previously was not leveraged, to provide more information to detection modules.

Let us revisit Example 2 to provide more intuition on our goal. The shortest string that reaches the critical state in the attacked system is $m_1m_{2i}m_1$. However, if we want to detect and prevent a successful attack, we must detect before it reaches the critical state. In this case, string $s = m_1m_{2i}$ is the shortest string where the attack can be mitigated since we can disable m_1 after detection at this point. The supervisor projection of s is $s_R = \Pi^R(s) = m_1m_2$ meanwhile its plant projection $s_H = \Pi^H(s) = m_1$. Thus, the supervisor observes string s_R while the plant executes string s_H. There are two options once string s_R is observed: it was genuinely generated by M_n or it was crafted by M_a. It is impossible to disambiguate these choices by only using string observation.

Let us compare the likelihood of s_R being generated in M_n with the likelihood of s being generated in M_a. The string s_R is generated by M_n with probability $0.1 * 0.\bar{1} = 0.0\bar{1}$, while s is generated by M_a with probability $0.1 * 1 = 0.1$. Thus, string s is 90% more likely to be generated than s_R, which means that it is more likely that M_a generated string s once string s_R is observed. This information can be used to disambiguate these strings. An intrusion detection module can make better decisions based on this new information.

In the next section, we introduce the notion of ϵ-safe systems which is related to the likelihood of system M_a being the generator of the aforementioned ambiguous strings. Intuitively, ϵ-safe systems are those with every ambiguous string being more likely to be generated by the attacked system M_a.

4.2 ϵ-safe systems

As was mentioned before, the intuition behind ϵ-safe systems is based on comparing the probability of generating a string before it reaches the critical state in M_a and the probability of its supervisor projection being generated in M_n. These two strings have the same observation and are considered *ambiguous* but it is possible that one is more likely to be generated than the other.

Similar as in the definition of safe-controllability (Paoli et al., 2011) and NA-safe-controlabillity (Lima et al., 2019), we say a state is a detection state if from this state the critical state is reachable by a string with uncontrollable events and one controllable event. We denote by X_{det} as the set of all detection states in M_a. Formally, $X_{det} = \{x \in X_{M_a} \mid \exists s \in (\Sigma_d \cup \Sigma)\Sigma_m^* \text{ s.t. } \delta_{M_a}(x, s) \in X_{crit,a} \land \Pi_H(s) \in \Sigma_c\Sigma_{uc}^*\}$.

Second, let $L_{det} = \{s \in \mathcal{L}(M_a) | (s_R \in \mathcal{L}(M_n)) \land (\delta_{M_a}(x_{0,M_a}, s) \in X_{det}) \land (\delta_{M_a}(x_{0,M_a}, s^i) \notin X_{det},\ i < |s|)\}$ be the set of ambiguous strings, i.e., strings in M_a that reach a state in the detection state set and whose supervisor projection is in M_n. Note that, L_{det} does not consider strings after a state in X_{det} is reached. In our running example, $X_{det} = \{\ 1'\}$ and $L_{det} = \{m_1m_{2i}, s_1m_1m_{2i}, s_2m_1m_{2i}, \dots\}$.

Definition 3. Given the two systems M_n and M_a and a fixed $\epsilon \in (0.5, 1]$, the system M_n is denoted as ϵ-safe with respect to an optimal reachability attacker if $\forall s \in L_{det}$, then

$$\frac{L_p(M_a)(s)}{L_p(M_n)(s_R) + L_p(M_a)(s)} \geq \epsilon \tag{5}$$

Definition 3 states that given an ambiguous string, the likelihood that this string is executed by M_a compared to the likelihood of its supervisor projection being executed by M_n is greater than ϵ. If we choose an ϵ value and M_n is ϵ-safe, then by observing an ambiguous string it is more likely, with at least ϵ

[2] The set $\tilde{L}_{crit,a}$ exactly contains independent measurable elements of $L_{crit,a}$.

confidence, that the attacked system generated this string. This definition was inspired by the maximum a posteriori probability (MAP) estimate.

It is important to note the similarity and the differences between our work and the work of fault diagnosis. In fault diagnosis of a logical system, the fault detector should be able to identify with certainty one that an observed string belongs to the faulty behavior. In the case of a probabilistic system, the notion of A-diagnosability states that in the limit the detector is able to identify the faulty behavior again with certainty one (Thorsley and Teneketzis, 2005; Bertrand et al., 2014; Yin et al., 2019). The notion of AA-diagnosability states that in the limit the failure event is included in the observed behavior with probability one but the failure detection decision is not made with certainty (Thorsley and Teneketzis, 2005; Bertrand et al., 2014; Yin et al., 2019).

Our definition of ϵ-safety does not distinguish the ambiguous strings with certainty one as in the case of diagnosability and A-diagnosability, nor distinguish these strings with an arbitrary small uncertainty as in the case of AA-diagnosability. It has a fixed parameter ϵ that defines the certainty level of the disambiguation of strings. The higher this parameter is the more certain the disambiguation becomes, e.g., when $\epsilon = 1$, we fall back into the intrusion detection of a logical system, which coincides with the definition of logical diagnosability.

Based on Definition 3, we state two verification problems.

Problem 1. Given a controlled system M_n and the set of compromised events Σ_a, verify if M_n is ϵ-safe w.r.t. an optimal reachability attacker.

Another problem is to find is the largest ϵ, if one exists, such that the controlled system M_n is ϵ-safe.

Problem 2. Given a controlled system M_n and the set of compromised events Σ_a, find, if it exists, $\epsilon^* = \inf\{\epsilon \in (0.5, 1] \mid M_n \text{ is } \epsilon\text{-safe}\}$.

5. SOLUTION METHODOLOGY

5.1 The verifier and solution intuition

Only strings in L_{det} are of interest, i.e., strings whose supervisor projections are in $\mathcal{L}(M_n)$ and that reach a detection state. We would like to construct an apparatus where these strings are easily manipulated. For this reason, we first construct the DFA T that captures strings in $\mathcal{L}(M_n) \cap \Pi^R(\mathcal{L}(M_a))$.

Definition 4. Let T be the DFA that generates $\Pi^R(\mathcal{L}(T)) = \mathcal{L}(M_n) \cap \Pi^R(\mathcal{L}(M_a))$. Namely, the states $X_T \in X_{M_n} \times X_{M_a}$, $x_{0,T} = (x_{0,M_n}, x_{0,M_a})$ and $\delta_T((x_1,x_2),e) = (y_1,y_2)$ if $\delta_{M_n}(x_1, \Pi^R(e)) = y_1$ and $\delta_{M_a}(x_2, e) = y_2$ for $e \in \Sigma_m$ and $(x_1,x_2),(y_1,y_2) \in X_{M_n} \times X_{M_a}$ with $x_2 \notin X_{det}$. The marked states of T are $X_{T,m} = \{(x_1,x_2) \in X_T \mid \exists s \in \mathcal{L}(T) \text{ s.t. } \delta_T(x_{0,T}, s) = (x_1,x_2),\ x_2 \in X_{det}\}$.

Note that, the marked states of T are related to the set L_{det}. The following lemma states this relationship.

Lemma 1. $\mathcal{L}_m(T) = L_{det}$.

Proof: It follows from the construction of T. □

Based on T, we define the verifier V that captures string executions of L_{det} in both M_n and M_a.

Definition 5. Based on $CoAc(T)$, we define the verifier $V = (X_V, \Sigma_m, P_V, x_{0,V}, X_{V,m})$. We have $X_V = X_{CoAc(T)}$, Σ_m is the complete set of events, $P_V : X_V \times \Sigma_m \times X_V \rightarrow [0,1]^2$, $x_{0,V} = x_{0,T}$ and $X_{V,m} = X_{T,m}$. The transition probability function P_V is defined as for $x = (x_1,x_2),\ y = (y_1,y_2) \in X_V$:

$$P_V(x,e,y) = \begin{cases} \begin{bmatrix} P_{M_n}(x_1, \Pi^R(e), y_1) \\ P_{M_a}(x_2, e, y_2) \end{bmatrix} & \text{if } \delta_{CoAc(T)}(x,e) = y \\ \text{undefined} & \text{otherwise} \end{cases}$$

Although the verifier V is not a PFA since P_V is defined differently, we apply definitions for PFA to the verifier V, e.g., $\mathcal{L}(V)$. In this manner, we state the following lemma.

Lemma 2. $\mathcal{L}_m(V) = \mathcal{L}_m(T)$ and $\mathcal{L}(V) = pre(\mathcal{L}_m(V))$.

Proof: It follows from the construction of V and Lemma 1. □

Given $x = (x_1,x_2),\ y = (y_1,y_2) \in X_V$ and $e \in \Sigma_m$, the first element of the vector $P_V(x,e,y)$ denotes the probability of generating $\Pi^R(e)$ in M_n from state $x_1 \in M_n$ to state $y_1 \in M_n$. The second element of $P_V(x,e,y)$ denotes the probability of generating event e in M_a from state $x_2 \in M_a$ to state $y_2 \in M_a$. Let $x_1 s[1] x_2 s[2] \ldots s[|s|] x_{|s|+1}$ be the subrun in V generated by $s \in \Sigma_m^*$ starting in state x_1. We define the vector $P_V^{s,x_1} = P_V(x_1, s[1], x_2) \odot P_V(x_2, s[2], x_3) \odot \cdots \odot P_V(x_{|s|}, s[|s|], x_{|s|+1})$, where $\odot$ is the entry-wise product of vectors. When $x_1 = x_{0,V}$, we use $P_V^s = P_V^{s,x_{0,V}}$.

We are now ready to state our main theorem on the verification of ϵ-safe systems.

Theorem 3. A system is ϵ-safe if and only if

$$\inf_{s \in \mathcal{L}_m(V)} \left\{ \frac{P_V^s[2]}{P_V^s[1] + P_V^s[2]} \right\} \geq \epsilon.$$

Proof: The proof follows from the Definition 3, Lemma 2, $L_p(M_n)(s_R) = P_V^s[1]$, and $L_p(M_a)(s) = P_V^s[2]$. □

What remains to be shown is the existence of an algorithm that checks the aforementioned condition. For that reason, we consider two cases:

(1) The set L_{det} has a finite number of strings. As a consequence, computing the ratio
$$\frac{P_V^s[2]}{P_V^s[1] + P_V^s[2]}$$
for all $s \in L_{det}$ can be completed in finite time. Moreover, based on Lemma 2, L_{det} has finite number of strings if and only if V is an acyclic directed graph.

(2) The set L_{det} has an infinite number of strings, which implies that V is cyclic. In this case, we cannot apply Theorem 3 directly as a test. We need to find another method for this case.

Next, we provide methods for both cases: acyclic and cyclic verifier V. Fortunately, these two cases are connected and the method used for an acyclic verifier is a special case of the general method for a cyclic verifier. Nonetheless, we present these two methods sequentially for readability purposes.

5.2 Acyclic verifier

Again, the set L_{det} has finitely many strings if and only if V is acyclic. There exists different algorithms to check if V is acyclic, e.g., Depth-First-Search, Tarjan's strongly connected components algorithm, etc. (Bang-Jensen and Gutin, 2008). We assume that the acyclicity of V has been confirmed and present Algorithm 1 to verify ϵ-safety. This algorithm simply checks each marked string in V individually for ϵ-safety.

Algorithm 1 Verification of ϵ-safe for finite L_{det}

```
Input: $V$ and $\epsilon$
Output: $\epsilon$-safety or not $\epsilon$-safe
function ACYCLIC_VERIFIER($V, \epsilon$)
    for all $s \in \mathcal{L}_m(V)$ do
        Compute $P_V^s$
        if $\frac{P_V^s[2]}{P_V^s[1]+P_V^s[2]} < \epsilon$ then
            return Not $\epsilon$-safe
        end if
    end for
    return $\epsilon$-safe
end function
```

We omit the proof of correctness of Algorithm 1 since it follows directly from Theorem 3 and the fact that V is acyclic.

5.3 Cyclic verifier

Although Theorem 3 holds when V is cyclic, the set of strings to be verified is infinite. Therefore, Algorithm 1 is a pseudo-algorithm since it does not terminate. Nevertheless, we will show that we can use the result for acyclic verifiers to handle the case of cyclic verifiers. Namely, we decompose the cyclic verifier into an acyclic verifier V_{ac} and a finite set of cycles C. Intuitively, we directly apply Algorithm 1 to V_{ac}. Algorithm 1 tests the ϵ-safety condition on V_{ac} but it is not sufficient to determine if V is ϵ-safe. To complete the test for V, we verify the set of cycles C in a similar manner as V_{ac}.

Proposition 4 provides the result on the existence of such decomposition.

Proposition 4. (Sect.1, pp-39 (Bang-Jensen and Gutin, 2008)) In a directed graph, every open walk (vertex can repeat) can be decomposed as a simple path (no repeated vertex) and simple cycles.

Proposition 4 is easily extended to DFA, where walks, paths and cycles are defined over runs and sub-runs $x_1 s[1] \dots x_{|s|} s[|s|]$ $x_{|s|+1}$.

We define V_{ac} as the acyclic part of V, i.e., one can think of V_{ac} as a rooted tree with $x_{0,V}$ as the root and states of $X_{V,m}$ as leaves. Namely, V_{ac} generates the marked language $\mathcal{L}_m(V_{ac}) = \{s \in \mathcal{L}_m(V) \mid \exists \text{ a run } x_1 s[1] x_2 s[2] \dots s[|s|] x_{|s|+1},\ x_1 = x_{0,V} \text{ and } x_i \neq x_j\ \forall i \neq j\}$. In this manner, Algorithm 1 can be directly applied to V_{ac}. Nonetheless, we only have a necessary condition since $\mathcal{L}_m(V_{ac}) \subseteq L_{det}$.

Next, we define C to be the set of all simple cycles in V, i.e., $C = \{x = x_1 s[1] \dots s[|s|] x_1 \in (X_V \Sigma_m)^* \mid x \text{ is a sub-run in V and } x_i \neq x_j,\ i \neq j \in [|s|]^+\}\}$. This set can be obtained by algorithms that find simple cycles in directed graphs, e.g., Johnson's algorithm (Mateti and Deo, 1976). The following theorem provides necessary and sufficient conditions to test ϵ-safety condition based on the constructed verifier V.

Theorem 5. Given the verifier V, the system M_n is ϵ-safe if and only if:

(1) The acyclic part of V denoted as V_{ac} is ϵ-safe; and
(2) For all $(x_1 s[1] \dots s[|s|] x_1) \in C$, $P_V^{s,x_1}[2] \geq P_V^{s,x_1}[1]$

Proof: We start by the only if part by assuming that M_n is ϵ-safe. The first condition is immediate since $\mathcal{L}_m(V_{ac}) \subseteq L_{det}$. We show the second condition by contradiction. Assume that there exists $c = y_1 t[1] y_2 \dots y_{|t|} t[|t|] y_1 \in C$ such that $P_V^{t,y_1}[1] > P_V^{t,y_1}[2]$. By Lemma 2 and the Pumping Lemma, there exists a run $x_1 s[1] x[2] \dots x_k s[k] c^n s[k+n|t|+1] x_{k+n|t|+1} \dots s[|s|] x_{|s|+1}$, where $x_1 = x_{0,V}$, $s = s_1 t^n s_2 = s[1] \dots s[k] t^n s[k+n|t|+1] \dots s[|s|] \in \mathcal{L}_m(V)$ and c^n means that the cycle sub-run is repeated n times. Since $s_1 t^n s_2 \in \mathcal{L}_m(V)$ and M_n is ϵ-safe:

$$\frac{P_V^{s_1 t^n s_2}[2]}{P_V^{s_1 t^n s_2}[1] + P_V^{s_1 t^n s_2}[2]} \geq \epsilon \tag{6}$$

$$\frac{P_V^{s_1 t^n s_2}[2]}{P_V^{s_1 t^n s_2}[1]} \geq \frac{\epsilon}{1-\epsilon} \tag{7}$$

Equation (7) is true for any $n \in \mathbb{N}$. We can rewrite $P_V^{s_1 t^n s_2}[i] = P_V^{s_1 s_2}[i] P_V^{t^n, y_1}[i]$, $i \in \{1,2\}$. From our assumption, $\frac{P_V^{t,y_1}[2]}{P_V^{t,y_1}[1]} < 1$. Thus, $\exists n_0 \in \mathbb{N}$ such that:

$$\frac{P_V^{s_1 s_2}[2]}{P_V^{s_1 s_2}[1]} \left(\frac{P_V^{t,y_1}[2]}{P_V^{t,y_1}[1]} \right)^{n_0} < \frac{\epsilon}{1-\epsilon} \tag{8}$$

It contradicts Equation (7) and the fact that M_n is ϵ-safe.

We move to the if part of the proof. It is assumed that conditions (1) and (2) are satisfied. Again Proposition 4 and Lemma 2 let us write any string $s \in L_{det}$ that generates the run $x_1 s[1] \dots s[|s|] x_{|s|+1}$ as $c_1 t[1] c_2 t[2] \dots c_{|t|} t[|t|] x_{|t|+1}$ where $t \in \mathcal{L}_m(V_{ac})$, $x_1 = x_{0,V}$, and c_i is a closed-run (starting and ending in x_i) or $c_i = x_i$ (no cyclic sub-run). The closed-run c_i is a composition of the simple cycles $c_i^1, \dots, c_i^{k_i}$ with $c_i^j = y_1^{ij} t_{ij}[1] y_2^{ij} \dots t_{ij}[|t_{ij}|] y_1^{ij} \in C$. This composition means that these simple cycles are used in some way, not necessarily sequentially, to construct c_i. Since $t \in \mathcal{L}_m(V_{ac})$, we have that

$$\frac{P_V^{t,x_1}[2]}{P_V^{t,x_1}[1]} \geq \frac{\epsilon}{1-\epsilon} \text{ (Condition (1))}$$

and

$$\frac{P_V^{t_{ij}, y_1^{ij}}[2]}{P_V^{t_{ij}, y_1^{ij}}[1]} \geq 1 \text{ (Condition (2))}$$

for every $i \in [|t|]^+$ and $j \in [k_i]^+$. As a consequence, we can group t with the composition of simple cycles to produce s and obtain

$$\frac{P_V^s[2]}{P_V^s[1]} \geq \frac{\epsilon}{1-\epsilon}$$

That concludes our proof. □

Theorem 5 shows that Problem 1 and Problem 2 are decidable since the sets $\mathcal{L}_m(V_{ac})$ and C are finite sets. Similar to Algorithm 1, we provide a general algorithm to verify the ϵ-safe condition of M_n.

The proof of correctness of Algorithm 2 follows directly from Theorem 5. Note that, Algorithm 2 has exponential complexity since both the number of finite strings in the marked language of V_{ac} and the number of cycles in C are exponential in the number of states in V.

Example 3. We return to our running example and verify if the controlled system M_n depicted in Fig. 2 is 0.9-safe w.r.t. the attacked system M_a illustrated in Fig. 3(b). We construct the verifier V, shown in Fig. 4. The acyclic verifier V_{ac} generates a marked language $\mathcal{L}_m(V_{ac}) = \{m_1 m_{2i}\}$. We see that V_{ac} is 0.9-safe. Next, the set of all single cycles in V is $C = \{(2,2)s_1(2,2), (2,2)s_2(2,2)\}$. It follows that condition (2) of Theorem 5 holds. Therefore, M_n is 0.9-safe w.r.t. M_a.

Algorithm 2 Verification of ϵ-safety for arbitrary L_{det}

```
Input: $V$ and $\epsilon$
Output: $\epsilon$-safe or not $\epsilon$-safe
Compute $V_{ac}$ and $C$ from $V$
if Not $\epsilon$-safe = ACYCLIC_VERIFIER($V_a, \epsilon$) then
    return Not $\epsilon$-safe
end if
for all $c = x_1 s[1] \dots s[|s|] x_{|s|+1} \in C$ do
    Compute $P_V^{s,x_1}$
    if $P_V^{s,x_1}[2] < P_V^{s,x_1}[1]$ then
        return Not $\epsilon$-safe
    end if
end for
return $\epsilon$-safe
```

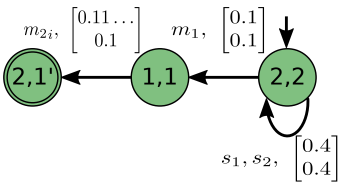

m_{2i}, $\begin{bmatrix} 0.11\dots \\ 0.1 \end{bmatrix}$
m_1, $\begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$
2,1'
1,1
2,2
s_1, s_2, $\begin{bmatrix} 0.4 \\ 0.4 \end{bmatrix}$

Fig. 4. Verifier V

6. CONCLUSION

We have considered the problem of detection of sensor deception attacks in the context of stochastic supervisory control theory. The notion of ϵ-safe systems is introduced as a first step to obtain quantitative measurements to help intrusion detection modules. These modules can use this information to reason about strings that are considered undetectable when only qualitative reasoning is used. Necessary and sufficient conditions to test ϵ-safety are presented. Furthermore, an algorithm is provided to test these conditions which shows that verification of ϵ-safety is decidable. The algorithms to test ϵ-safety presented in this paper are in the worst case exponential in time.

The presented definition of ϵ-safety is parameterized by a specific attack strategy. It would be of interest to consider a more general definition parameterized by a class of attack strategies (e.g., deterministic attack strategies). Moreover, it would be of interest to soften the ϵ-safety condition.

REFERENCES

Bang-Jensen, J. and Gutin, G.Z. (2008). *Digraphs: Theory, Algorithms and Applications*. Springer Publishing Company, Incorporated, 2nd edition.

Bertrand, N., Haddad, S., and Lefaucheux, E. (2014). Foundation of Diagnosis and Predictability in Probabilistic Systems. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 29, 417–429. New Delhi, India.

Cardenas, A.A., Amin, S., and Sastry, S. (2008). Secure control: Towards survivable cyber-physical systems. In *2008 The 28th International Conference on Distributed Computing Systems Workshops*, 495–500.

Carvalho, L.K., Wu, Y.C., Kwong, R., and Lafortune, S. (2018). Detection and mitigation of classes of attacks in supervisory control systems. *Automatica*, 97, 121 – 133.

Cassandras, C.G. and Lafortune, S. (2008). *Introduction to Discrete Event Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2 edition.

Garg, V.K., Kumar, R., and Marcus, S.I. (1999). A probabilistic language formalism for stochastic discrete-event systems. *IEEE Transactions on Automatic Control*, 44(2), 280–293.

Kumar, R. and Garg, V.K. (2001). Control of stochastic discrete event systems modeled by probabilistic languages. *IEEE Transactions on Automatic Control*, 46(4), 593–606.

Lawford, M. and Wonham, W.M. (1993). Supervisory control of probabilistic discrete event systems. In *Proceedings of 36th Midwest Symposium on Circuits and Systems*, 327–331.

Lima, P.M., Alves, M.V.S., Carvalho, L.K., and Moreira, M.V. (2019). Security against communication network attacks of cyber-physical systems. *Journal of Control, Automation and Electrical Systems*, 30(1), 125–135.

Mateti, P. and Deo, N. (1976). On algorithms for enumerating all circuits of a graph. *SIAM J. Comput.*, 5, 90–99.

Meira-Góes, R., Kang, E., Kwong, R., and Lafortune, S. (2017). Stealthy deception attacks for cyber-physical systems. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, 4224–4230.

Meira-Góes, R., Kang, E., Kwong, R., and Lafortune, S. (2019a). Synthesis of sensor deception attacks at the supervisory layer of cyber-physical systems. *under review*.

Meira-Góes, R., Kwong, R., and Lafortune, S. (2019b). Synthesis of sensor deception attacks for systems modeled as probabilistic automata. In *2019 American Control Conference (ACC)*.

Meira-Góes, R., Marchand, H., and Lafortune, S. (2019c). Towards resilient supervisors against sensor deception attacks. In *To appear at 2019 IEEE 58th Annual Conference on Decision and Control (CDC)*.

Pantelic, V., Lawford, M., and Postma, S. (2014). A framework for supervisory control of probabilistic discrete event systems. *12th IFAC International Workshop on Discrete Event Systems (WODES)*, 47(2), 477 – 484.

Paoli, A., Sartini, M., and Lafortune, S. (2011). Active fault tolerant control of discrete event systems using online diagnostics. *Automatica*, 47(4), 639–649.

Rashidinejad, A., Wetzels, B., Reniers, M., Lin, L., Zhu, Y., and Su, R. (2019). Supervisory control of discrete-event systems under attacks: An overview and outlook. In *2019 18th European Control Conference (ECC)*, 1732–1739.

Su, R. (2018). Supervisor synthesis to thwart cyber attack with bounded sensor reading alterations. *Automatica*, 94, 35 – 44.

Teixeira, A., Pérez, D., Sandberg, H., and Johansson, K.H. (2012). Attack models and scenarios for networked control systems. In *Proceedings of the 1st International Conference on High Confidence Networked Systems*, HiCoNS '12, 55–64. ACM, New York, NY, USA.

Thorsley, D. and Teneketzis, D. (2005). Diagnosability of stochastic discrete-event systems. *IEEE Transactions on Automatic Control*, 50(4), 476–492.

Thorsley, D. and Teneketzis, D. (2006). Intrusion detection in controlled discrete event systems. In *Proceedings of the 45th IEEE Conference on Decision and Control*, 6047–6054.

Wang, Z., Meira-Góes, R., Lafortune, S., and Kwong, R. (2020). Mitigation of classes of attacks using a probabilistic discrete event system framework. In *15th IFAC Workshop on Discrete Event Systems WODES 2020 (to appear)*.

Yin, X., Chen, J., Li, Z., and Li, S. (2019). Robust fault diagnosis of stochastic discrete event systems. *IEEE Transactions on Automatic Control*, 64, 4237–4244.