



# Non-Deterministic Liveness-Enforcing Supervisor Tolerant to Sensor-Reading Modification Attacks

Dan You , *Member, IEEE*, and Shouguang Wang , *Senior Member, IEEE*

**Abstract**—In this paper, we study the supervisory control problem of discrete event systems assuming that cyber-attacks might occur. In particular, we focus on the problem of liveness enforcement and consider a sensor-reading modification attack (SM-attack) that may disguise the occurrence of an event as that of another event by intruding sensor communication channels. To solve the problem, we introduce non-deterministic supervisors in the paper, which associate to every observed sequence a set of possible control actions offline and choose a control action from the set randomly online to control the system. Specifically, given a bounded Petri net (PN) as the reference formalism and an SM-attack, an algorithm that synthesizes a liveness-enforcing non-deterministic supervisor tolerant to the SM-attack is proposed for the first time.

**Index Terms**—Cyber-attacks, cyber-physical system (CPS), liveness, non-deterministic supervisors, Petri net (PN).

## I. INTRODUCTION

A cyber-physical system (CPS) integrates computing and communication to monitor and control physical processes [1]. The use of communication networks not only endows the physical components of CPS with information processing and communication capabilities, but also increases the vulnerability of CPS to cyber-attacks [2]. The supervisory control of CPS under cyber-attacks has been receiving more and more attentions, especially in the area of discrete event systems (DESs). By modeling a CPS as a DES, the existing studies dealing with attack issues can be classified into three categories, namely, attack strategy design [3]–[5], attack detection and defense [6]–[10], and robust/tolerant supervisor design [3], [11]–[19]. In this work, we investigate the design of robust/tolerant supervisors in the sense that a given control specification can be satisfied by such a supervisor despite the existence of attacks.

Much attention has been paid to the robust/tolerant supervisor design in the literature. Wakaiki *et al.* [11] design a super-

visor robust against the so-called replacement-removal attacks in the sensor channels. In more detail, the attacker may erase an event produced by the plant or replace it with another one by tampering with sensor-readings in related sensor channels. Essentially, their work is similar to the work dealing with the supervisory control of DES with nondeterministic observations [13]–[15]. The conditions under which a supervisor exists exactly enforcing a given control specification are provided in [13], [14], while a supervisor synthesis method is given in [15] using a model transformation technique. Su [3] considers sensor deception attacks as well but requires attacks to be stealthy, which means that the supervisor cannot be aware of the existence of attacks. Specifically, the attacker may replace an observable event generated by the plant with a bounded sequence of observable events. From the viewpoint of an attacker, the supremal stealthy attack strategy is computed first and then a supervisor robust against the attack is synthesized from the viewpoint of a system defender. Meira-Góes *et al.* [12] investigate the synthesis of supervisors robust to parameterized sensor deception attacks. In addition to sensor deception attacks, actuator deception attacks are taken into account in [16]–[18].

The above studies basically consider language or state specifications. In contrast, the liveness-enforcing control specification has received little attention in the presence of attacks. Liveness is a property characterizing a specific dynamic behavior of a system. How to enforce liveness has been widely studied in different problem settings [20]–[23]. Nevertheless, to our best knowledge, [19] is the only work dealing with the liveness-enforcing problem under attacks. In [19], the problem is investigated based on bounded Petri net (PN) systems [24], [25] and sensor-reading modification attacks (SM-attacks) are considered, which are a particular class of the replacement-removal attacks in [11]. In the scenario of SM-attacks, an intruder has the ability to disguise the occurrence of some transitions as that of other transitions by modifying sensor-readings in vulnerable sensor channels. An algorithm that synthesizes a liveness-enforcing supervisor tolerant to SM-attacks is provided in [19]. Unfortunately, the supervisor is not guaranteed to be maximally permissive. How to get a maximally permissive liveness-enforcing supervisor tolerant to an SM-attack remains an open issue.

We note that the supervisor in [19] refers to a deterministic supervisor, which means that the supervisor associates every observed sequence with a unique control action. In contrast to deterministic supervisors, there are non-deterministic supervisors. In simple words, a non-deterministic supervisor asso-

Manuscript received February 1, 2023; revised April 6, 2023 and May 8, 2023; accepted May 24, 2023. This work was supported in part by the Public Technology Research Plan of Zhejiang Province (LGJ21F030001), the National Natural Science Foundation of China (62302448), and the Zhejiang Provincial Key Laboratory of New Network Standards and Technologies (2013 E10012). Recommended by Associate Editor Jiachun Wang. (*Corresponding author: Shouguang Wang.*)

Citation: D. You and S. Wang, “Non-deterministic liveness-enforcing supervisor tolerant to sensor-reading modification attacks,” *IEEE/CAA J. Autom. Sinica*, vol. 11, no. 1, pp. 240–248, Jan. 2024.

The authors are with the School of Information and Electronic Engineering, Sussex Artificial Intelligence Institute, Zhejiang Gongshang University, Hangzhou 310018, China (e-mail: youdan@zjgsu.edu.cn, wangshouguang@zjgsu.edu.cn).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/JAS.2023.123702

ciates every observed sequence with a set of possible control actions. Which one is chosen to control the system is determined randomly online. Non-deterministic supervisors are first proposed to tackle the standard supervisory control problem under partial observation [26], [27]. Compared with using deterministic supervisors, the main advantages of using them include that 1) The complexity of synthesizing a supervisor is reduced from being exponential to polynomial; and 2) More control specifications can be realized such as temporal logic CTL\* [28] and  $\mu$ -calculus [29]. Moreover, in the setting of the non-deterministic plant and non-deterministic control specification, non-deterministic supervisors are capable of solving a (bi)similarity control problem [29]–[33]. Recently, they are also used to solve an opacity enforcing problem [34].

In this paper, we design non-deterministic supervisors to enforce the property of liveness on DES under SM-attacks. As far as we know, the liveness-enforcing problem under attacks has never been solved by using non-deterministic supervisors. Indeed, in the scenario of SM-attacks, it could happen that an observation does not explicitly reflect real information. For example, in the case that an SM-attack may disguise the firing of transition  $t_1$  as the firing of transition  $t_2$ , when we observe  $t_2$ , we are not sure which transition fires producing the observation  $t_2$  if  $t_1$  and  $t_2$  are both enabled. It is the vague information that makes the liveness-enforcing problem under SM-attacks very complex. The use of a non-deterministic supervisor can exactly eliminate the vague situation and keep more possible behavior of the controlled system. In the above example, a non-deterministic supervisor may give two possible control actions, namely, disabling  $t_1$  or  $t_2$ . No matter which one is chosen online to control the system, we know with certainty which transition fires producing the observation  $t_2$ . Following this basic idea, we develop an algorithm in the paper, which synthesizes a liveness-enforcing non-deterministic supervisor tolerant to an SM-attack. We may see that, compared with the current methods that synthesize a deterministic supervisor, the proposed method enjoys lower computational complexity in synthesizing a non-deterministic one. Note that as in [19] PNs are used as a modeling tool to solve the problem in this paper and only bounded PN systems are considered.

The remainder of the paper is organized as follows. Section II reviews the basic notions involved in the paper. Section III formulates the problem to be addressed in the paper. How to synthesize a non-deterministic liveness-enforcing supervisor tolerant to SM-attacks is answered in Section IV. Discussion on the proposed method is provided in Section V. Section VI concludes this paper and indicates future work.

## II. PRELIMINARIES

### A. Petri Nets

A Petri net (PN) is a quadruple  $N = (P, T, F, W)$ , where  $P$  is the set of places,  $T$  is the set of transitions,  $F \subseteq (P \times T) \cup (T \times P)$  is the flow relation of the net, and  $W: F \rightarrow \mathbb{Z}^+$  assigns to each arc a weight. Given a node  $x \in P \cup T$ ,  $\cdot x$  denotes the set of inputs of  $x$ , i.e.,  $\cdot x = \{x' \in P \cup T \mid (x', x) \in F\}$ , and  $x \cdot$  denotes the set of outputs of  $x$ , i.e.,  $x \cdot = \{x' \in P \cup T \mid (x, x') \in F\}$ . A marking

of  $N$  is a mapping  $m: P \rightarrow \mathbb{N}$ .  $(N, m_0)$  is called a PN system with initial marking  $m_0$ . A transition  $t \in T$  is enabled at a marking  $m$ , denoted as  $m[t]$ , if  $m(p) \geq W(p, t)$ ,  $\forall p \in \cdot t$ . If the firing of the enabled transition  $t$  at marking  $m$  reaches marking  $m'$ , it is written as  $m[t]m'$ . A sequence of transitions  $\sigma = t_1 t_2 \dots t_k \in T^*$  is enabled at  $m$ , denoted as  $m[\sigma]$ , if there exist markings  $m_1, m_2, \dots, m_{k-1}$  such that  $m[t_1]m_1[t_2]m_2[t_3] \dots m_{k-1}[t_k]$ . If the firing of the enabled transition sequence  $\sigma$  at marking  $m$  reaches marking  $m'$ , it is written as  $m[\sigma]m'$ . The set of all reachable markings of  $(N, m_0)$  is defined as  $R(N, m_0) = \{m \mid \exists \sigma \in T^*, \text{ s.t. }, m_0[\sigma]m\}$ . A transition  $t$  is live at a marking  $m$  if  $\forall m' \in R(N, m)$ ,  $\exists m'' \in R(N, m')$ ,  $m''[t]$ . A transition  $t$  is dead at a marking  $m$  if  $\forall m' \in R(N, m)$ ,  $t$  is disabled at  $m'$ . A net system  $(N, m_0)$  is live if  $\forall t \in T$ ,  $t$  is live at  $m_0$ . The language of  $(N, m_0)$ , denoted by  $L(N, m_0)$ , is defined as the set of all sequences that are enabled at the initial marking  $m_0$ , i.e.,  $L(N, m_0) = \{\sigma \in T^* \mid m_0[\sigma]\}$ . A PN system  $(N, m_0)$  is bounded if  $\exists B \in \mathbb{N}^+ = \{1, 2, \dots\}$ , s.t.  $\forall m \in R(N, m_0)$ ,  $\forall p \in P$ ,  $m(p) \leq B$ . Otherwise, it is unbounded.

A reachability graph (RG) of a PN system  $(N, m_0)$  is a directed graph describing the system evolution. Specifically, each node of the RG corresponds to a reachable marking  $m \in R(N, m_0)$  and each arc is labelled by a transition  $t \in T$ . Specifically, there exists an arc labelled by transition  $t$  from a node representing marking  $m_1$  to a node representing marking  $m_2$  in the RG if and only if it holds that  $m_1[t]m_2$ .

### B. Deterministic and Non-Deterministic Supervisors

Given a PN system  $(N, m_0)$ , we use  $L_o(N, m_0)$  to denote the set of all possible observed sequences of the system and use  $\Delta = 2^T$  to denote the set of all control actions, each of which refers to a set of transitions to be disabled, or equivalently, whose firing should be forbidden.

A deterministic supervisor of a PN system  $(N, m_0)$  is  $\mu: L_o(N, m_0) \rightarrow \Delta$ , associating to every observed sequence a control action. A non-deterministic supervisor of a PN system  $(N, m_0)$  is  $\mu: L_o(N, m_0) \rightarrow 2^\Delta$ , associating to every observed sequence a set of control actions. It is worth noting that only one control action is chosen among the set to control the PN system during its evolution and it is chosen online non-deterministically.

Given a PN system  $(N, m_0)$  and a supervisor  $\mu$ , we denote the system under the control of  $\mu$  as  $(N, m_0)|_\mu$  and the set of reachable markings of  $(N, m_0)|_\mu$  as  $R(N, m_0)|_\mu$ .

A supervisor  $\mu$  is said to be *liveness-enforcing* for a PN system  $(N, m_0)$  if the controlled system  $(N, m_0)|_\mu$  is live, i.e.,  $\forall t \in T$ ,  $\forall m \in R(N, m_0)|_\mu$ ,  $\exists m' \in R(N, m)|_\mu$ , s.t.  $m'[_\mu t]$ , where  $m'[_\mu t]$  denotes that  $t$  can fire at marking  $m'$  under the control of  $\mu$ .

### C. SM-Attacks

We define an SM-attack as a mapping  $O: T \rightarrow 2^T$ .

The set  $O(t)$  enumerates all the possible observations when  $t$  occurs due to the existence of the SM-attack.

*Example 1:* Consider a bounded PN  $(N, m_0)$  with the set of transitions  $T = \{t_1, t_2, t_3\}$  and an SM-attack  $O$  such that  $O = [O(t_1), O(t_2), O(t_3)]^T = [\{t_1\}, \{t_2, t_3\}, \{t_3\}]^T$ . We can see that,

due to the existence of the SM-attack, the occurrence of  $t_2$  might produce the observation  $t_2$  or  $t_3$ . In other words, the occurrence of  $t_2$  is possibly disguised as the occurrence of  $t_3$ . As for  $t_1$  and  $t_3$ , their observations can never be changed.

We may generalize the notation  $O(\cdot)$  to a sequence of transitions  $\sigma = t_1 t_2 \dots t_k \in T^*$  such that  $O(\sigma) = O(t_1)O(t_2) \dots O(t_k)$ . Note that given two sets  $A$  and  $B$ , it is defined that  $AB = \{ab \mid a \in A, b \in B\}$ . Then, given a PN system  $(N, m_0)$  and an SM-attack  $O$ , we have  $L_o(N, m_0) = \bigcup_{\sigma \in L(N, m_0)} O(\sigma)$ .

### III. PROBLEM STATEMENT

The problem that we plan to address in the paper is formulated as follows.

**Problem 1:** Given a bounded PN system  $(N, m_0)$  and an SM-attack  $O$ , design a liveness-enforcing supervisor tolerant to  $O$ .

“Tolerant” means that the supervisor is liveness-enforcing even in the presence of an SM-attack. The problem has been investigated in [19], where the solution is a deterministic supervisor that is not guaranteed to be maximally permissive. In this work, we intend to solve the problem by designing a supervisor that is allowed to be non-deterministic.

### IV. NON-DETERMINISTIC CONTROL

In this section, we first introduce a so-called supervisor graph that represents a supervisor in this paper and then provide an algorithm that synthesizes a non-deterministic supervisor for solving Problem 1.

#### A. Supervisor Graph

A structure called a supervisor graph is formally defined in [19] to describe a deterministic supervisor intuitively. We still use it in this paper but with some modification such that it can describe a non-deterministic supervisor as well. In simple words, a supervisor graph is now a directed graph where a node is assigned to be an initial node, each arc is labelled by a transition and each node is associated with a set of control actions. Note that we simply write  $\mu(x)$  to represent the set of control actions associated with a node  $x$ . Indeed, for all observed sequences leading from the initial node to node  $x$  in a supervisor graph, they are associated with the same set of control actions, i.e.,  $\mu(x)$ .

**Example 2:** Fig. 1 shows a supervisor graph with an initial node  $x_0$ . It tells that 1) for any observed sequence leading from initial node  $x_0$  to node  $x_1$ , the set of control actions is  $\{\{t_2\}, \{t_3\}\}$ , which means that it is chosen non-deterministically online to disable  $t_2$  or  $t_3$ ; 2) for any observed sequence leading from  $x_0$  to  $x_3$ ,  $t_1$  should be disabled; and 3) for any observed sequence leading from  $x_0$  to node  $x_0$  or  $x_2$ , no transition should be disabled. To be intuitive, nodes with a non-empty set of control actions are colored in grey.

In the remainder of the paper, we omit writing  $\mu(x) = \emptyset$  for every node  $x$  in a supervisor graph to save space. Besides, a node in a supervisor graph basically corresponds to a set of markings. Thus, for the sake of simplicity, we may name a node by its corresponding marking set if there is no ambiguity.

**Example 3:** Consider a PN system  $(N, m_0)$  in Fig. 2(a). Its

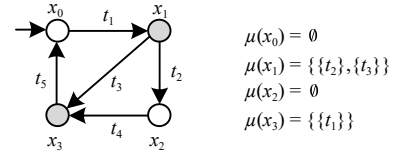


Fig. 1. Supervisor graph.

RG  $G_r$  shown in Fig. 2(b) can be viewed as a supervisor graph of  $(N, m_0)$ . It tells that whatever we observe, we do not disable any transition. Also, graph  $G_l$  in Fig. 3 is a supervisor graph of  $(N, m_0)$ . It indicates that 1) When we observe a sequence (e.g.,  $t_2$ ) leading to node  $\{m_1\}$ , we should disable  $t_2$ ; 2) When we observe a sequence (e.g.,  $t_2 t_3$ ) leading to node  $\{m_3\}$ , we should disable  $t_4$ ; and 3) When we observe a sequence (e.g.,  $t_3$ ) leading to other nodes, we do not disable any transition. Trivially,  $G_l$  represents a deterministic supervisor since no node is associated with a set containing two or more control actions.

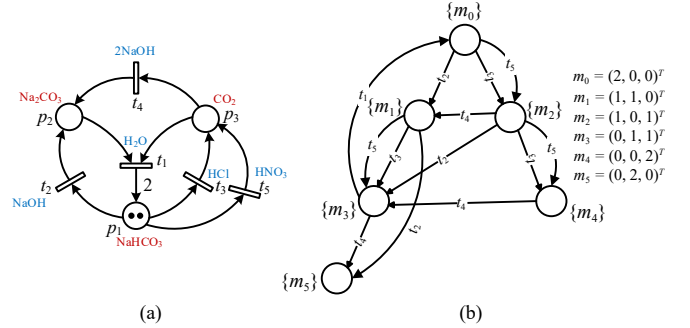


Fig. 2. Case study. (a) PN system  $(N, m_0)$  modelled for chemical reactions; (b) Its RG  $G_r$ .

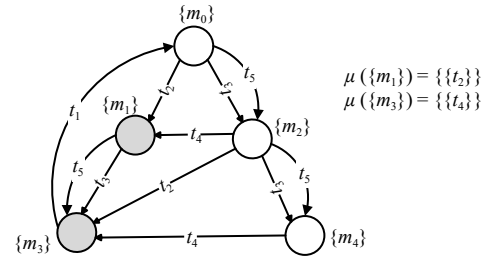


Fig. 3. Maximally permissive liveness-enforcing deterministic supervisor  $G_l$  of the PN system  $(N, m_0)$  in Fig. 2(a) under no attack.

We conclude this subsection by introducing some structures in a supervisor graph. Given a supervisor graph  $G$ , we say  $\alpha = x_1 l_1 x_2 l_2 \dots l_{n-1} x_n$  a path if  $\forall i \in \{1, 2, \dots, n-1\}$ , there exists a directed arc with label  $l_i$  from node  $x_i$  to node  $x_{i+1}$  in  $G$ . Alternatively, we may write the path  $\alpha = x_1 l_1 x_2 l_2 \dots l_{n-1} x_n$  as  $x_1 \xrightarrow{l_1} x_2 \xrightarrow{l_2} \dots \xrightarrow{l_{n-1}} x_n$ . Given two nodes  $x_1$  and  $x_2$  in  $G$ , we say  $x_2$  is accessible from  $x_1$  (or equivalently,  $x_1$  can access  $x_2$ ) if there exists a path from  $x_1$  to  $x_2$ . By default, a node is accessible from itself via an empty path. A subgraph  $G'$  of  $G$  is said to be a strongly connected component (SCC) if  $G'$  is strongly connected and maximal. We use  $La(G')$  to denote the set of arc labels that appear in SCC  $G'$ . Given a node  $x$  and an SCC

$G'$  in  $G$ , we say that  $G'$  is accessible from  $x$  (or equivalently,  $x$  can access  $G'$ ) if there exists a node  $x'$  in  $G'$  accessible from  $x$ . Note that  $x$  and  $x'$  can be the same node and we can see that if a node can access an SCC, the node can access every node in this SCC. Moreover, given a node  $x$  in a supervisor graph, we may use  $\cdot x$  to denote the set of input nodes of  $x$  and  $x\cdot$  to denote the set of output nodes of  $x$ . A node  $x$  is said to be a sink node if it has no output node, i.e.,  $x\cdot = \emptyset$ . An SCC  $G'$  is said to be a sink SCC if no node outside  $G'$  is accessible from a node in  $G'$ .

### B. Proposed Algorithm

In this subsection, we propose an algorithm to solve Problem 1. To this end, we introduce the following concepts.

**Definition 1:** Given a set  $T' \subseteq T$  of transitions and an SM-attack  $O$ , we say that  $\Pi \subseteq 2^{T'}$  is a feasible partition on  $T'$  w.r.t.  $O$  if

- 1)  $\cup_{\pi \in \Pi} \pi = T'$ ;
- 2)  $\forall \pi \in \Pi, |\pi| \geq 1$ ; and
- 3)  $\forall \pi \in \Pi, (|\pi| = 1) \vee (\forall t_1, t_2 \in \pi, O(t_1) \cap O(t_2) = \emptyset)$ .

In Definition 1, condition 1) means that the union of all sets in  $\Pi$  is equal to the set  $T'$ ; condition 2) requires that every set in  $\Pi$  contains at least one transition; and condition 3) indicates that for every set in  $\Pi$ , either it contains only one transition or no two transitions in it have the same observation.

**Definition 2:** Given a set  $T' \subseteq T$  of transitions and an SM-attack  $O$ , a feasible partition  $\Pi \subseteq 2^{T'}$  is said to be optimal if for any other feasible partition  $\Pi' \subseteq 2^{T'}$ , it holds that  $\forall \pi \in \Pi, \forall \pi' \in \Pi', \pi \not\subseteq \pi'$ .

In simple words, when we compute a feasible partition on a set of transitions, we actually divide the set of transitions into several groups such that any two transitions in a group do not have the same observation. The optimality of a feasible partition indicates that no more transition can be added to any group such that the partition remains to be feasible.

We next present a function named *OptimalFeasiblePartition* (i.e., Function 1), by which the optimal feasible partition is derived. Note that, given a transition  $t \in T$  and an SM-attack  $O$ , we denote  $\tilde{T}(t) = \{t' \in T \setminus \{t\} \mid O(t) \cap O(t') \neq \emptyset\}$ , namely,  $\tilde{T}(t)$  consists of all transitions who possibly produce the same observation as  $t$  due to the SM-attack  $O$ .

---

#### Function 1 $\Pi := \text{OptimalFeasiblePartition}(T', O)$

---

**Input:** A set  $T' \subseteq T$  of transitions and an SM-attack  $O$ ;

**Output:**  $\Pi \subseteq 2^{T'}$ .

- 1)  $\Pi := \{T'\}$ ;
  - 2) **for**  $t \in T'$  **do**
  - 3)   **for**  $\pi \in \Pi$  **do**
  - 4)     **if**  $t \in \pi \wedge \tilde{T}(t) \cap \pi \neq \emptyset$  **then**
  - 5)        $\pi_1 := \pi \setminus \tilde{T}(t)$  and  $\pi_2 := \pi \setminus \{t\}$ ;
  - 6)        $\Pi := \Pi \setminus \{\pi\} \cup \{\pi_1, \pi_2\}$ ;
  - 7)     **end if**
  - 8)   **end for**
  - 9) **end for**
  - 10) **output:**  $\Pi$ .
- 

**Result 1:** Given a set of transitions  $T' \subseteq T$  and an SM-attack

$O$ ,  $\Pi := \text{OptimalFeasiblePartition}(T', O)$  is the optimal feasible partition on  $T'$  w.r.t.  $O$ .

*Proof:* By contradiction, suppose that  $\Pi$  is not the optimal feasible partition. It means that there exists a feasible partition  $\Pi'$  such that  $\exists \pi \in \Pi, \exists \pi' \in \Pi', \pi \subset \pi'$ . Let  $t \in \pi \setminus \pi'$ . By function *OptimalFeasiblePartition*,  $\exists t' \in \pi$ , s.t.  $t \in \tilde{T}(t')$  since otherwise  $t$  is a transition in  $\pi$ . Hence, we can see that  $\exists t, t' \in \pi',$  s.t.,  $t \in \tilde{T}(t')$ , which contradicts that  $\Pi'$  is a feasible partition. As a result,  $\Pi$  is the optimal feasible partition on  $T'$  w.r.t.  $O$ . ■

**Example 4:** Consider a PN system  $(N, m_0)$  with the set of transitions  $T = \{t_1 - t_7\}$  and an SM-attack  $O = [O(t_1), O(t_2), O(t_3), O(t_4), O(t_5), O(t_6), O(t_7)]^T = [\{t_1, t_6, t_7\}, \{t_2, t_6\}, \{t_3, t_7\}, \{t_4, t_3\}, \{t_5\}, \{t_6\}, \{t_7\}]^T$ . Consider a set of transitions  $T' = \{t_1 - t_5\}$ . We compute the optimal feasible partition on  $T'$  by calling *OptimalFeasiblePartition* ( $T', O$ ). First,  $\Pi$  is initialized as  $\{\pi\}$ , where  $\pi = \{t_1 - t_5\}$ . Then, we consider  $t_1$ . Since  $\tilde{T}(t_1) = \{t_2, t_3, t_6, t_7\}$ , we have  $\pi_1 = \pi \setminus \tilde{T}(t_1) = \{t_1, t_4, t_5\}$  and  $\pi_2 = \pi \setminus \{t_1\} = \{t_2, t_3, t_4, t_5\}$ . Hence,  $\Pi$  is updated as  $\{\pi_1, \pi_2\}$ . Similarly, we consider  $t_2, t_3, t_4$ , and  $t_5$  one after another. Finally, we get the set  $\Pi = \{\{t_1, t_4, t_5\}, \{t_2, t_3, t_5\}, \{t_2, t_4, t_5\}\}$ , which is the optimal feasible partition on  $T'$  w.r.t.  $O$ .

Now, we are ready to introduce Algorithm 1. Note that, given a node  $x$  in a supervisor graph  $G$ , we use  $T_{\text{out}}^G(x)$  to denote the set of transitions labelling the output arcs of  $x$  in  $G$ . Moreover, node  $x$  is said to be a confusing node w.r.t. an SM-attack  $O$  if  $\exists t_1, t_2 \in T_{\text{out}}^G(x)$ , s.t.  $O(t_1) \cap O(t_2) \neq \emptyset$ . Also, we may call a supervisor graph as a supervisor for the sake of simplicity in the following discussions.

---

#### Algorithm 1 Supervisor Synthesizer

---

**Input:** A bounded PN system  $(N, m_0)$  and an SM-attack  $O$ .

**Output:** A supervisor graph  $G^\#$ .

- 1)  $G_l := \text{MaxLiveEnforceSupervisor}(N, m_0)$ ; /\*See Function 2.
  - 2) *MaxLiveEnforceSupervisor* returns a maximally permissive liveness-enforcing deterministic supervisor of  $(N, m_0)$  under no attack [19] \*/
  - 3) **for** each node  $x$  in  $G_l$  **do**
  - 4)   **if**  $x$  is a confusing node w.r.t.  $O$  **then**
  - 5)      $\Pi := \text{OptimalFeasiblePartition}(T_{\text{out}}^{G_l}(x), O)$ ;
  - 6)      $\mu(x) := \cup_{\pi \in \Pi} \{(T_{\text{out}}^{G_l}(x) \setminus \pi) \cup \mu(x)\}$ ;
  - 7)   **end if**
  - 8)   **for** each  $t \in T_{\text{out}}^{G_l}(x)$  **do**
  - 9)     **for**  $t' \in O(t) \setminus \{t\}$  **do**
  - 10)       add an arc labelled by  $t't$  from  $x$  to  $x'$ , where  $x'$  is a node reached from node  $x$  via an arc labelled by  $t$ ;
  - 11)     **end for**
  - 12)   **end for**
  - 13) denote the final graph as  $G^\#$ ;
  - 14) **output:**  $G^\#$ .
- 

---

#### Function 2 $G_l = \text{MaxLiveEnforceSupervisor}(N, m_0)$

---

**Input:** A bounded PN system  $(N, m_0)$  with  $N = (P, T, F)$ ;

**Output:** A maximally permissive liveness-enforcing deterministic supervisor  $G_l$ .

- 1) compute the RG  $G_r$  of  $(N, m_0)$ ;
- 2)  $\Phi := \text{Tarjan}(G_r)$ ; /\*Tarjan [35] here returns the set of SCCs excluding those that are exactly a single node in  $G_r$ \*/
- 3)  $\Phi^\# := \{\alpha \in \Phi \mid T \notin La(\alpha)\}$ ;
- 4)  $G_l := G_r$ ;
- 5) **while** there exists a sink node  $x$  or a sink SCC  $\alpha \in \Phi^\#$  in  $G_l$  **do**
- 6)   **if** there exists a sink node  $x$  in  $G_l$  **then**
- 7)     **for** each  $x' \in \cdot x$  **do**
- 8)        $\mu(x') := \mu(x') \cup T'$ , where  $T'$  denotes the set of transitions labeling the arcs from  $x'$  to  $x$ ;
- 9)   **end for**
- 10)   update  $G_l$  by deleting  $x$  and its related arcs;
- 11)   **end if**
- 12)   **if** there exists a sink SCC  $\alpha \in \Phi^\#$  in  $G_l$  **then**
- 13)     **for** each  $x' \in \cdot x_\alpha$ , where  $x_\alpha$  is a node in  $\alpha$  and  $x'$  is a node outside  $\alpha$  **do**
- 14)        $\mu(x') := \mu(x') \cup T'$ , where  $T'$  denotes the set of transitions labeling the arcs from  $x'$  to  $x_\alpha$ ;
- 15)     **end for**
- 16)     update  $G_l$  by deleting  $\alpha$  and its related arcs;
- 17)   **end if**
- 18) **end while**
- 19) **output:**  $G_l$ .

We explain Algorithm 1 as follows. First, we compute a maximally permissive liveness-enforcing deterministic supervisor for the input PN system under no attack. The supervisor can be computed by calling  $G_l = \text{MaxLiveEnforceSupervisor}(N, m_0)$ , which is adapted from [19]. Next, for every node  $x$  in  $G_l$ , we determine if it is a confusing node, i.e., if there are at least two transitions in  $T_{\text{out}}^{G_l}(x)$  who possibly produce the same observation due to the attack  $O$ . If so, we update the set of control actions at the node. Specifically, we first compute the optimal feasible partition  $\Pi$  on  $T_{\text{out}}^{G_l}(x)$  w.r.t.  $O$  by Step 4 and then get a set of control actions (i.e.,  $\mu(x)$ ) based on  $\Pi$  by Step 5. The purpose of computing a feasible partition on  $T_{\text{out}}^{G_l}(x)$  w.r.t.  $O$  is to divide transitions in  $T_{\text{out}}^{G_l}(x)$  into several groups such that any two transitions in a group cannot have the same observation under the attack. In this way, if we permit the firing of transitions in one group, we know with certainty which transition actually fires when getting an observation. Note that, when computing a feasible partition on  $T_{\text{out}}^{G_l}(x)$ , we consider the control in terms of permitting the firing of some transitions, while a control action defined in the paper is to forbid the firing of some transitions. Thus, we do the equivalent transformation as shown in Step 5 to determine  $\mu(x)$ . Specifically, for each group  $\pi \in \Pi$ ,  $T_{\text{out}}^{G_l}(x) \setminus \pi$  is accordingly a set of transitions to be disabled. Also, the set of transitions disabled originally at node  $x$  should be taken into consideration for computing each control action. Moreover, we note that the computation of the optimal feasible partition aims to improve the permissiveness of each control action. Then, we add some output arcs to node  $x$  (no matter if it is a confusing node) such that all the possible observations under attack  $O$  are included, which is shown in Steps 7–11. This indeed leads to the resulting supervisor with two kinds of arc labels. We

note that a label with two transitions, e.g.,  $t|t'$ , denotes that  $t'$  is observed but what actually occurred is  $t$ , while a label with a single transition, e.g.,  $t$ , indicates that  $t$  is observed and what actually occurred is also  $t$ . The final supervisor graph output by Algorithm 1 is denoted as  $G^\#$ .

How  $G^\#$  works online to control the PN system  $(N, m_0)$  under attack  $O$  is shown exhaustively in Procedure 1. In Procedure 1, we use  $x_{\text{cur}}$  to denote the current node reached in the supervisor  $G^\#$  during the system evolution and the current control action on the PN system is chosen non-deterministically among  $\mu(x_{\text{cur}})$ . At the beginning,  $x_{\text{cur}}$  is initialized as the initial node  $x_0$  of  $G^\#$  and thereby we choose non-deterministically among  $\mu(x_0)$  a set  $T_{\text{dis}}$  of transitions to disable. Then, every time we observe a transition in the PN system, we first determine which node is reached accordingly in the supervisor  $G^\#$ , i.e., update  $x_{\text{cur}}$  and then choose non-deterministically among  $\mu(x_{\text{cur}})$  a set  $T_{\text{dis}}$  of transitions to disable.

---

**Procedure 1** Online Execution of Supervisor  $G^\#$

---

**Input:** A bounded PN system  $(N, m_0)$  under an SM-attack  $O$  and the supervisor  $G^\#$  computed by Algorithm 1

- 1)  $x_{\text{cur}} := x_0$ ; /\*  $x_0$  is the initial node of  $G^\#$  \*/
  - 2) choose non-deterministically among  $\mu(x_{\text{cur}})$  a set of transitions to disable, denoted as  $T_{\text{dis}}$ ;
  - 3) **while** a transition  $t$  is observed **do**
  - 4)   find an output arc of  $x_{\text{cur}}$  with a label  $l$  such that 1)  $l = t \wedge t \notin T_{\text{dis}}$ ; or 2)  $l = t|t' \wedge t' \notin T_{\text{dis}}$ ;
  - 5)    $x_{\text{cur}} := \text{Reach}(x_{\text{cur}}, l)$ ; /\*  $\text{Reach}(x_{\text{cur}}, l)$  returns a node reached from node  $x_{\text{cur}}$  via an output arc with label  $l$  \*/
  - 6)   choose non-deterministically among  $\mu(x_{\text{cur}})$  a set of transitions to disable, denoted as  $T_{\text{dis}}$ ;
  - 7) **end while**
- 

*Example 5:* Continue to consider Example 4. Suppose that the maximally permissive liveness-enforcing deterministic supervisor  $G_l$  of the PN system  $(N, m_0)$  under no attack is computed and there is a node  $x$  in  $G_l$  such that  $T_{\text{out}}^{G_l}(x) = T' = \{t_1 - t_5\}$  and  $\mu(x) = \{\{t_6\}\}$ . To be intuitive, we consider a case as depicted in Fig. 4(a) where only the subgraph of  $G_l$  that consists of node  $x$  together with its output arcs and nodes is shown. Clearly,  $x$  is a confusing node. Let us see how Algorithm 1 works regarding  $x$ . First, we compute the optimal feasible partition  $\Pi$  on  $T_{\text{out}}^{G_l}(x)$  w.r.t.  $O$ , which has been derived in Example 4. Next, it is computed that  $\mu(x) = \{\{t_2, t_3, t_6\}, \{t_1, t_4, t_6\}, \{t_1, t_3, t_6\}\}$ . Finally, we add output arcs to node  $x$ , which results in the subgraph in Fig. 4(b). This subgraph tells that a control action at node  $x$  is chosen non-deterministically among  $\{t_2, t_3, t_6\}$ ,  $\{t_1, t_4, t_6\}$  and  $\{t_1, t_3, t_6\}$ . Consider as an example that  $\{t_2, t_3, t_6\}$  is chosen as the control action and then  $t_6$  is observed. By Procedure 1, we should select the arc with label  $t_6|t_1$ , which means that it is the firing of  $t_1$  that produces the observation  $t_6$ , and thus we reach node  $x_1$  to determine the next control action.

Given a bounded PN system  $(N, m_0)$ , an SM-attack  $O$ , and a supervisor  $G^\#$  computed by Algorithm 1, we have the following result.



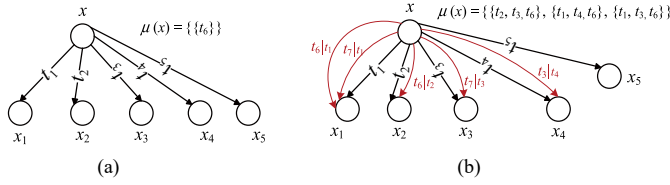


Fig. 4. Illustrative example: (a) A subgraph of  $G_I$ ; (b) A subgraph of  $G^\#$  computed by Algorithm 1.

**Lemma 1:** Let  $x$  and  $x'$  be two nodes in  $G^\#$  corresponding to markings  $m$  and  $m'$ , respectively. It holds that  $m' \in R(N, m)|_{G^\#}$  if there exists a path from node  $x$  to node  $x'$  in  $G^\#$ .

*Proof:* Without loss of generality, we assume that the path from node  $x$  to node  $x'$  in  $G^\#$  is

$$x_1 \xrightarrow{l_1} x_2 \xrightarrow{l_2} x_3 \xrightarrow{l_3} \dots \xrightarrow{l_{n-2}} x_{n-1} \xrightarrow{l_{n-1}} x_n$$

where  $x_i$  ( $i = 1, 2, \dots, n$ ) is a node with  $x_1 = x$  and  $x_n = x'$  and  $l_i$  ( $i = 1, 2, \dots, n-1$ ) is the arc label. Moreover, let  $m_1, m_2, \dots, m_n$  be the markings associated with nodes  $x_1, x_2, \dots, x_n$  in  $G^\#$ , respectively. Note that  $m_1 = m$  and  $m_n = m'$ . We observe that, for every node  $x$  in  $G^\#$ ,  $\mu(x)$  is computed based on a feasible partition on  $T_{\text{out}}^{G_I}(x)$ . It thus holds that, for  $i = 1, 2, \dots, n$ ,  $\exists T_{\text{dis}(i)} \in \mu(x_i)$  such that 1)  $l_i = t \wedge t \notin T_{\text{dis}(i)}$ ; or 2)  $l_i = t' \wedge t \notin T_{\text{dis}(i)}$ . In either case,  $m_{i+1}$  is reachable from  $m_i$  in the case that a control action is appropriately chosen among  $\mu(x_i)$ . Consequently, we have  $m_n \in R(N, m_1)|_{G^\#}$  i.e.,  $m' \in R(N, m)|_{G^\#}$ . ■

**Theorem 1:** Given a bounded PN system  $(N, m_0)$  and an SM-attack  $O$ ,  $G^\#$  computed by Algorithm 1 is a liveness-enforcing supervisor tolerant to  $O$  if  $G^\# \neq \emptyset$ .

*Proof:* The proof consists of two parts, namely, 1)  $G^\#$  is a feasible supervisor under attack  $O$  in the sense that  $G^\#$  associates every observed sequence a set of control actions; and 2)  $G^\#$  is liveness-enforcing.

1) Algorithm 1 first computes a liveness-enforcing deterministic supervisor  $G_I$  under no attack. It means that  $G_I$  associates to every observed sequence, which is also the really occurred sequence, a control action. Structurally,  $G^\#$  is a resultant graph by adding more arcs to  $G_I$ . Since arcs are added in detail as Steps 7–11 of Algorithm 1 for every node in  $G_I$ , all the possible observations and their corresponding really fired transitions in the presence of attack  $O$  are provided in  $G^\#$ . Moreover, at every confusing node  $x$  in  $G_I$  (i.e.,  $\exists t_1, t_2 \in T_{\text{out}}^{G_I}(x)$ , s.t.  $O(t_1) \cap O(t_2) \neq \emptyset$ ), we update  $\mu(x)$ , the set of control actions. Since  $\mu(x)$  is computed based on the optimal feasible partition on  $T_{\text{out}}^{G_I}(x)$  w.r.t.  $O$ , whatever control action is chosen from  $\mu(x)$ , we know with certainty which transition actually fires when getting an observation. In other words, when we observe a transition at  $x$ , we can find one and only one arc that leads from the node to the next node by Procedure 1. Thus, for every observed sequence, it corresponds to one and only one set of control actions in  $G^\#$ . As a result,  $G^\#$  is a feasible supervisor under attack  $O$ .

2) For the sake of simplicity, we say that an SCC  $G'$  in a supervisor graph is complete if its arc labels cover all transitions of the considered PN, i.e.,  $T \subseteq La(G')$ . Now, we start the proof as follows.

Since  $G_I$  is a liveness-enforcing deterministic supervisor under no attack, we have  $\forall t \in T, \forall m \in R(N, m_0)|_{G_I}, \exists m' \in R(N, m)|_{G_I}$ , s.t.  $m' \xrightarrow{t}$ . Moreover, every node in  $G_I$  can access a complete SCC since  $(N, m_0)$  is a bounded system. Structurally, every node in  $G^\#$  can also access a complete SCC since  $G^\#$  is a resultant graph by adding more arcs to  $G_I$ .

Let  $x$  be a node in  $G^\#$  corresponding to a marking  $m$ . Clearly, there exists a path in  $G^\#$  from the initial node  $x_0$  to  $x$ . Thus, we have  $m \in R(N, m_0)|_{G^\#}$  due to Lemma 1. Also,  $x$  can access a complete SCC. Hence, for each  $t \in T$ , there exists a node  $x'$  with a marking  $m'$  in the complete SCC such that  $m' \in R(N, m)|_{G^\#}$  and  $\exists T_{\text{dis}} \in \mu(x')$ , s.t.  $t \notin T_{\text{dis}}$ , i.e.,  $t$  can fire at  $m'$ . Trivially, all the markings in  $G^\#$  constitute the set  $R(N, m_0)|_{G^\#}$ . As a result, we can see that

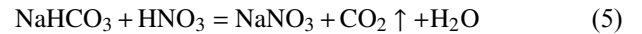
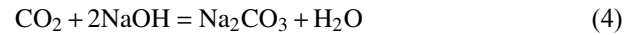
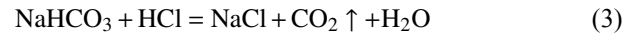
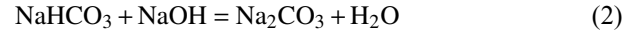
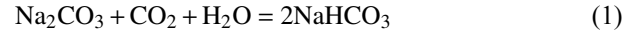
$$\forall t \in T, \forall m \in R(N, m_0)|_{G^\#}, \exists m' \in R(N, m)|_{G^\#}, \text{ s.t. } m' \xrightarrow{t}.$$

Therefore,  $G^\#$  is liveness-enforcing. ■

### C. Case Study

In this subsection, we provide a case study where the proposed method is applied. This case study is originally presented in [19].

An automatic control system is designed for playing a game on chemical reactions, where chemical (1)–(5) are involved.



Suppose that NaOH, HCl, HNO<sub>3</sub> and H<sub>2</sub>O are sufficiently provided and the chemical transformation among Na<sub>2</sub>CO<sub>3</sub>, CO<sub>2</sub>, and NaHCO<sub>3</sub> is the focus of the game. Consequently, the chemical reactions can be modelled by the PN system in Fig. 2(a), where  $p_1$ – $p_3$  represent NaHCO<sub>3</sub>, Na<sub>2</sub>CO<sub>3</sub> and CO<sub>2</sub>, respectively, and  $t_1$ – $t_5$  represent the chemical reactions (1)–(5), respectively. To be intuitive, the meanings of places and the needed substances triggering chemical transformation are annotated. Place  $p_1$  contains two tokens in the initial marking, which models the initial condition that two copies of NaHCO<sub>3</sub> are provided. The control specification on the system requires that any of the five chemical (1)–(5) can always be performed after finite times of other chemical reactions. This corresponds to the liveness specification on the PN system in Fig. 2(a).

The supervisor in the closed-loop control system observes the occurrence of chemical reactions by receiving signals from sensors and gives control actions based on observations. The supervisor communicates with sensors/actuators via communication networks, which makes the system possibly suffer from the intrusion of malicious agents. Suppose that we have the prior knowledge that the sensor signals produced by the chemical (2) are prone to be disguised as the sensor signals produced by the chemical (3) in related sensor communication channels by an intruder. Then, we can see that the system

is vulnerable to the SM-attack  $O = [O(t_1), O(t_2), O(t_3), O(t_4), O(t_5)]^T = [\{t_1\}, \{t_2, t_3\}, \{t_3\}, \{t_4\}, \{t_5\}]^T$ . In other words, the SM-attack may disguise the occurrence of  $t_2$  as the occurrence of  $t_3$ .

Now, we compute a liveness-enforcing supervisor tolerant to the SM-attack for the PN system in Fig. 2(a) by Algorithm 1. First, we compute a maximally permissive liveness-enforcing deterministic supervisor  $G_l$  of  $(N, m_0)$  under no attack, which is shown in Fig. 3. Next, we update the set of control actions at every confusing node in  $G_l$  and add more arcs to  $G_l$ . There are two confusing nodes, i.e.,  $\{m_0\}$  and  $\{m_2\}$ . Consider node  $\{m_0\}$ . The optimal feasible partition on  $T_{out}^{G_l}(\{m_0\})$  w.r.t.  $O$  is  $\Pi = \{\{t_2, t_5\}, \{t_3, t_5\}\}$  and thus we have  $\mu(\{m_0\}) = \{\{t_3\}, \{t_2\}\}$ . Then, we add an arc labelled by  $t_3|t_2$  from node  $\{m_0\}$  to node  $\{m_1\}$ . Similarly, for node  $\{m_2\}$ , we have  $\mu(\{m_2\}) = \{\{t_3\}, \{t_2\}\}$  and add an arc labelled by  $t_3|t_2$  from node  $\{m_2\}$  to node  $\{m_3\}$ . Since no more arcs should be added, we obtain the final supervisor  $G^\#$  as shown in Fig. 5. It is a liveness-enforcing non-deterministic supervisor for the PN system in Fig. 2(a) tolerant to  $O$ .

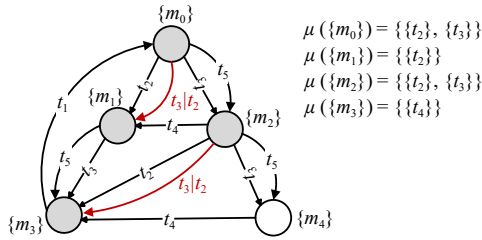


Fig. 5. Supervisor  $G^\#$  computed by Algorithm 1 in the case study.

## V. DISCUSSION

We can see that a non-deterministic supervisor, if associating to every observed sequence a set with only one control action, is essentially a deterministic supervisor; while in other cases, a non-deterministic supervisor can be viewed as the combination of multiple or even countless deterministic supervisors. The online execution of a non-deterministic supervisor is indeed also a procedure of determining a deterministic supervisor that is used to control the system since a control action is fixed online every time a sequence is observed.

Consider the non-deterministic supervisor in Fig. 5. It can be viewed as the combination of countless deterministic supervisors. In particular, there are two choices at node  $\{m_0\}$  and  $\{m_2\}$ , namely, disabling  $t_2$  or  $t_3$ . In the case that we always disable  $t_2$  at node  $\{m_0\}$  and disable  $t_3$  at node  $\{m_2\}$ , we may extract a deterministic supervisor, as shown in Fig. 6, from the non-deterministic supervisor. The supervisor in Fig. 6 is indeed a maximally permissive liveness-enforcing deterministic supervisor tolerant to the SM-attack  $O$  in the case study. Note that a liveness-enforcing deterministic supervisor is also computed by the method in [19] for the case study, which is shown in Fig. 7. We can see that the supervisor in Fig. 7 is more restrictive than the one in Fig. 6, i.e., it is not maximally permissive. For example, sequence  $t_3 t_5$  is allowed

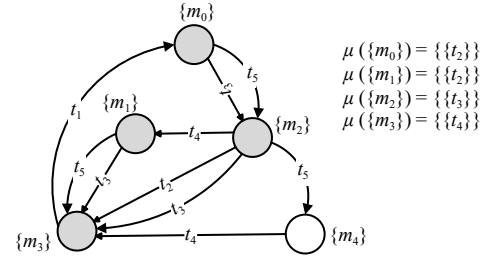


Fig. 6. Maximally permissive liveness-enforcing deterministic supervisor tolerant to the SM-attack  $O$  in the case study.

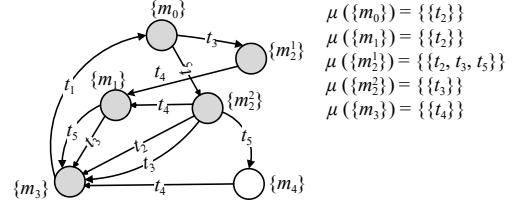


Fig. 7. Liveness-enforcing deterministic supervisor tolerant to the SM-attack  $O$  in the case study computed by the method in [19].

to fire by the supervisor in Fig. 6, while it is not allowed to fire by the supervisor in Fig. 7. On the contrary, sequences that are allowed to fire by the supervisor in Fig. 7 are all allowed to fire by the supervisor in Fig. 6.

We finally compare the computational complexity of the proposed method and the one in [19] in finding a solution to Problem 1. We observe that both methods start from constructing a maximally permissive liveness-enforcing deterministic supervisor  $G_l$  under no attack. This step determines the exponential complexity of both methods with respect to the size of the considered PN system since the RG has to be computed. We then focus on their remaining steps for comparison in what follows. Consider the proposed method. In the worst case, it updates once the set of control actions at every node of  $G_l$  and handles every arc of  $G_l$  once. Suppose that  $G_l$  contains  $a$  nodes and arcs in total. Then, the complexity of the proposed method is  $O(a)$  in getting a solution. In other words, the proposed method is of polynomial complexity with respect to the net size of  $G_l$ . Consider the method in [19]. It has to repeatedly compute a so-called basic supervisor, which is of exponential complexity with respect to the net size of  $G_l$  [19]. Consequently, we may conclude that the proposed method has lower computational complexity than the method in [19] in synthesizing a supervisor. In particular, when a maximally permissive liveness-enforcing deterministic supervisor  $G_l$  under no attack is given, the complexity of getting a solution is reduced from the exponential one to the polynomial one by using the proposed method. We note that the only additional cost of using non-deterministic supervisors lies in choosing a control action randomly online. Fortunately, such computation is actually negligible.

## VI. CONCLUSIONS AND FUTURE WORK

In this paper, we consider the use of non-deterministic con-

trol mechanisms to solve the liveness-enforcing problem in the presence of cyber-attacks. Specifically, for a bounded Petri net system vulnerable to a sensor-reading modification attack (SM-attack), an algorithm is proposed that synthesizes an SM-attack-tolerant liveness-enforcing supervisor that is allowed to be non-deterministic. Compared with the method in [19] synthesizing a liveness-enforcing deterministic supervisor tolerant to an SM-attack, the proposed method is simpler in computational complexity. In our future work, we may consider the improvement of the proposed method so that the resulting supervisor is more permissive. Its extension to unbounded Petri nets [36]–[38] remains open.

## REFERENCES

- [1] R. Baheti and H. Gill, “Cyber-physical systems,” *The Impact of Control Technology*, vol. 12, no. 1, pp. 161–166, 2011.
- [2] J. Zhang, L. Pan, Q.-L. Han, C. Chen, S. Wen, and Y. Xiang, “Deep learning based attack detection for cyber-physical system cybersecurity: A survey,” *IEEE/CAA J. Autom. Sinica*, vol. 9, no. 3, pp. 377–391, 2021.
- [3] R. Su, “Supervisor synthesis to thwart cyber attack with bounded sensor reading alterations,” *Automatica*, vol. 94, pp. 35–44, 2018.
- [4] R. Meira-Góes, E. Kang, R. H. Kwong, and S. Lafortune, “Synthesis of sensor deception attacks at the supervisory layer of cyber-physical systems,” *Automatica*, vol. 121, p. 109172, 2020.
- [5] L. Lin and R. Su, “Synthesis of covert actuator and sensor attackers,” *Automatica*, vol. 130, p. 109714, 2021.
- [6] D. Thorsley and D. Teneketzis, “Intrusion detection in controlled discrete event systems,” in *Proc. 45th IEEE Conf. Decision and Control*, 2006, pp. 6047–6054.
- [7] L. K. Carvalho, Y.-C. Wu, R. Kwong, and S. Lafortune, “Detection and mitigation of classes of attacks in supervisory control systems,” *Automatica*, vol. 97, pp. 121–133, 2018.
- [8] P. M. Lima, M. V. S. Alves, L. K. Carvalho, and M. V. Moreira, “Security against communication network attacks of cyber-physical systems,” *J. Control, Automation and Electrical Systems*, vol. 30, no. 1, pp. 125–135, 2019.
- [9] M. Agarwal, S. Purwar, S. Biswas, and S. Nandi, “Intrusion detection system for PS-Poll DoS attack in 802.11 networks using real time discrete event system,” *IEEE/CAA J. Autom. Sinica*, vol. 4, no. 4, pp. 792–808, 2017.
- [10] M. Agarwal, S. Biswas, and S. Nandi, “Discrete event system framework for fault diagnosis with measurement inconsistency: Case study of rogue DHCP attack,” *IEEE/CAA J. Autom. Sinica*, vol. 6, no. 3, pp. 789–806, 2019.
- [11] M. Wakaiki, P. Tabuada, and J. P. Hespanha, “Supervisory control of discrete-event systems under attacks,” *Dynamic Games and Applications*, vol. 9, pp. 965–983, 2019.
- [12] R. Meira-Góes, S. Lafortune, and H. Marchand, “Synthesis of supervisors robust against sensor deception attacks,” *IEEE Trans. Automatic Control*, vol. 66, no. 10, pp. 4990–4997, Oct. 2021.
- [13] T. Ushio and S. Takai, “Nonblocking supervisory control of discrete event systems modeled by mealy automata with nondeterministic output functions,” *IEEE Trans. Autom. Control*, vol. 61, no. 3, pp. 799–804, 2016.
- [14] S. Xu and R. Kumar, “Discrete event control under nondeterministic partial observation,” in *Proc. IEEE Int. Conf. Automation Science and Engineering*, 2009, pp. 127–132.
- [15] X. Yin, “Supervisor synthesis for mealy automata with output functions: A model transformation approach,” *IEEE Trans. Autom. Control*, vol. 62, no. 5, pp. 2576–2581, 2017.
- [16] L. Lin, Y. Zhu, and R. Su, “Towards bounded synthesis of resilient supervisors,” in *Proc. IEEE 58th Conf. Decision and Control*, 2019, pp. 7659–7664.
- [17] Y. Wang and M. Pajic, “Attack-resilient supervisory control with intermittently secure communication,” in *Proc. IEEE 58th Conf. Decision and Control*, 2019, pp. 2015–2020.
- [18] R. Meira-Góes, H. Marchand, and S. Lafortune, “Dealing with sensor and actuator deception attacks in supervisory control,” *Automatica*, vol. 147, p. 110736, 2023.
- [19] D. You, S. Wang, and C. Seatzu, “A liveness-enforcing supervisor tolerant to sensor-reading modification attacks,” *IEEE Trans. Systems, Man, and Cyber.: Syst.*, vol. 52, no. 4, pp. 2398–2411, 2022.
- [20] C. Chen and H. Hu, “Liveness-enforcing supervision in AMS-oriented hamgs: An approach based on new characterization of siphons using Petri nets,” *IEEE Trans. Autom. Control*, vol. 63, no. 7, pp. 1987–2002, 2017.
- [21] Y. Chen and Z. Li, “Design of a maximally permissive liveness-enforcing supervisor with a compressed supervisory structure for flexible manufacturing systems,” *Automatica*, vol. 47, no. 5, pp. 1028–1034, 2011.
- [22] B. Huang, M. Zhou, Y. Huang, and Y. Yang, “Supervisor synthesis for fms based on critical activity places,” *IEEE Trans. Syst., Man, and Cyber.: Syst.*, vol. 49, no. 5, pp. 881–890, 2019.
- [23] D. You, S. Wang, and M. Zhou, “Synthesis of monitor-based liveness-enforcing supervisors for S3PR with  $\xi$ -resources,” *IEEE Trans. Syst., Man, and Cyber.: Syst.*, vol. 45, no. 6, pp. 967–975, 2015.
- [24] G. Liu, *Petri Nets: Theoretical Models and Analysis Methods for Concurrent Systems*. Singapore: Springer 2022.
- [25] L. He and G. Liu, “Prioritized time-point-interval Petri nets modelling multi-processor real-time systems and TCTL<sub>q</sub>,” *IEEE Trans. Industrial Informatics*, vol. 19, no. 8, pp. 8784–8794, 2023.
- [26] K. Inan, “Nondeterministic supervision under partial observations,” in *Proc. 11th Int. Conf. Analysis and Optimization of Syst. Discrete Event Syst.*, 1994, pp. 39–48.
- [27] R. Kumar, S. Jiang, C. Zhou, and W. Qiu, “Polynomial synthesis of supervisor for partially observed discrete-event systems by allowing nondeterminism in control,” *IEEE Trans. Autom. Control*, vol. 50, no. 4, pp. 463–475, 2005.
- [28] S. Jiang and R. Kumar, “Supervisory control of discrete event systems with CTL\* temporal logic specifications,” *SIAM J. Control and Optimization*, vol. 44, no. 6, pp. 2079–2103, 2006.
- [29] S. Basu and R. Kumar, “Control of non-deterministic systems with calculus specifications using quotienting,” *IEEE/CAA J. Autom. Sinica*, vol. 8, no. 5, pp. 953–970, 2021.
- [30] M. Fabian and B. Lennartson, “On non-deterministic supervisory control,” in *Proc. 35th IEEE Conf. Decision and Control*, 1996, vol. 2, pp. 2213–2218.
- [31] H. Farhat, “Control of nondeterministic systems for bisimulation equivalence under partial information,” *IEEE Trans. Autom. Control*, vol. 65, no. 12, pp. 5437–5443, 2020.
- [32] C. Zhou, R. Kumar, and S. Jiang, “Control of nondeterministic discrete-event systems for bisimulation equivalence,” *IEEE Trans. Autom. Control*, vol. 51, no. 5, pp. 754–765, 2006.
- [33] S. Takai, “Synthesis of maximally permissive supervisors for nondeterministic discrete event systems with nondeterministic specifications,” *IEEE Trans. Autom. Control*, vol. 66, no. 7, pp. 3197–3204, 2021.
- [34] Y. Xie, X. Yin, and S. Li, “Opacity enforcing supervisory control using non-deterministic supervisors,” *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 1763–1769, 2020.
- [35] R. Tarjan, “Depth-first search and linear graph algorithms,” *SIAM J. Computing*, vol. 1, no. 2, pp. 146–160, 1972.
- [36] J. Li, X. Yu, and M. Zhou, “Analysis of unbounded Petri net with lean reachability trees,” *IEEE Trans. Syst., Man, and Cybern.: Syst.*, vol. 50, no. 6, pp. 2007–2016, 2020.



- [37] F. Lu, Q. Zeng, M. Zhou, Y. Bao, and H. Duan, "Complex reachability trees and their application to deadlock detection for unbounded Petri nets," *IEEE Trans. Syst., Man, and Cyber.: Syst.*, vol. 49, no. 6, pp. 1164–1174, 2019.
- [38] S. Wang, M. Gan, M. Zhou, and D. You, "A reduced reachability tree for a class of unbounded Petri nets," *IEEE/CAA J. Autom. Sinica*, vol. 2, no. 4, pp. 345–352, 2015.



**Dan You** (Member, IEEE) received the B.S. degree in electronic and information engineering and the M.S. degree in information and communication engineering from the School of Information and Electronic Engineering, Zhejiang Gongshang University in 2014 and 2017, respectively, and the Ph.D. degree in electronic and computer engineering from the Department of Electrical and Electronic Engineering, the University of Cagliari, Italy in 2021. She is currently a Member of the Discrete-Event System Group, the School of Information and Electronic Engineering, Zhejiang Gongshang University. Her research interests include supervisory control of

discrete event systems, fault prediction, and deadlock control and siphon computation in Petri nets.



**Shouguang Wang** (Senior Member, IEEE) received the B.S. degree in computer science from the Changsha University of Science and Technology in 2000, and the Ph.D. degree in electrical engineering from Zhejiang University in 2005. He joined Zhejiang Gongshang University in 2005, where he is currently a Professor with the School of Information and Electronic Engineering, the Director of the Discrete-Event Systems Group, and the Dean of System modeling and Control Research Institute, Zhejiang Gongshang University. He was a Visiting Professor with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, USA, from 2011 to 2012. He is a Visiting Professor with the Electrical and Electronic Engineering Department, University of Cagliari, Italy, from 2014 to 2015. He was the Dean of the Department of Measuring and Control Technology and Instrument from 2011 to 2014. His research interests include supervisory control of discrete event systems and siphon computation in Petri nets. He is currently an Associate Editor of the *IEEE/CAA Journal of Automatica Sinica*.