Networked Supervisory Control Synthesis of Timed Discrete-Event Systems

Citation for published version (APA):
Rashidinejad, A., Reniers, M. A., & Fabian, M. (2021). Networked Supervisory Control Synthesis of Timed Discrete-Event Systems. *arXiv*, *2021*, Article 2102.09255. https://doi.org/10.48550/arXiv.2102.09255

DOI:
10.48550/arXiv.2102.09255

Document status and date:
Published: 18/02/2021

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Networked Supervisory Control Synthesis of Timed Discrete-Event Systems

Aida Rashidinejad Michel Reniers Martin Fabian

***Abstract*—Conventional supervisory control theory assumes full synchronization between the supervisor and the plant. This assumption is violated in a networked-based communication setting due to the presence of delays, and this may result in incorrect behavior of a supervisor obtained from conventional supervisory control theory. This paper presents a technique to synthesize a networked supervisor handling communication delays. For this purpose, first, a networked supervisory control framework is provided, where the supervisor interacts with the plant through control and observation channels, both of which introduce delays. The control channel is FIFO, but the observation channel is assumed to be non-FIFO so that the observation of events may not necessarily be received by the supervisor in the same order as they occurred in the plant. It is assumed that a global clock exists in the networked control system, and so the communication delays are represented in terms of time. Based on the proposed framework, a networked plant automaton is achieved, which models the behavior of the plant under the effects of communication delays and disordered observations. Based on the networked plant, the networked supervisor is synthesized, which is guaranteed to be (timed networked) controllable, nonblocking, time-lock free, (timed networked) maximally permissive, and satisfies control requirements for the plant.**

***Index Terms*—Discrete-event systems, time delays, networked control, maximal permissiveness, nonblockingness, safety, supervisory control, synthesis.**

I. Introduction

NETWORKED control of systems has gained a lot of attention in recent years. By eliminating unnecessary wiring, the cost and complexity of a control system are reduced, and nodes can more easily be added to or removed from the system. More importantly, there are applications in which the system is required to be controlled over a distance such as telerobotics, space explorations, and working in hazardous environments [1].

Networked control of systems is challenging due to network communication problems among which delays have the highest impact [2]. In this regard, many works have appeared in the literature investigating the effects of communication delays on the performance of a control system with time-based dynamics [1]–[3]. However, there is less work considering networked control of discrete-event systems (DESs).

A DES consists of a set of discrete states where state transitions depend only on the occurrence of instantaneous events. DESs are used for modeling many types of systems, e.g., manufacturing processes, traffic, and queuing systems [4]. In DESs, time is typically neglected meaning that events can occur independently of time. However, there are control applications in which time is an important factor to be considered, such as minimizing the production-cycle time in a manufacturing process [5]. To consider time in control of a DES, the concept of a timed discrete-event system (TDES) has been introduced, in which the passage of a unit of time is indicated by an event called *tick* [6].

Supervisory control theory is the main control approach developed for DESs [7]. To achieve desired (safe) behavior, a supervisor observes events executed in the plant and determines which of the next possible events must be disabled. Supervisory control theory synthesizes nonblocking supervisors that ensure safety, controllability, and nonblockingness for the plant and do not unnecessarily restrict the behavior of the plant (maximal permissiveness) [4].

In conventional supervisory control theory [4], [8], the plant generates all events, while the supervisor can disable some of the events and observes synchronously the execution of events in the plant. Based on this synchronous interaction, a model of the controlled plant behavior can be obtained by synchronous composition of the respective models of the plant and the supervisor. However, the synchronous interaction assumption fails in a networked supervisory control setting, due to the presence of delays in the communication channels between the plant and supervisor.

There are several works in the literature investigating supervisory control of DES under communication delays. There are three important properties that these works may focus on:

1) Nonblockingness. For many applications, it is important to guarantee that the supervised plant does not block (as an additional control requirement) [4], [9], [10].

2) Maximal permissiveness. A supervisor must not restrict the plant behavior more than necessary so that the maximal admissible behavior of the plant is preserved [4], [5].

3) Timed delays (delays modeled based on time). In most of the existing approaches such as in [9], [11]–[17], communication delays are measured in terms of a number of consecutive event occurrences. As stated in [15], [18], [19], it is not proper to measure time delay only based on the number of event occurrences since events may have different execution times. Here, as in TDES [5], the event *tick* is used to represent the passage of a unit of time, which is the temporal resolution for modeling purposes.

This research has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under grant agreement no 674875.

Aida Rashidinejad and Michel Reniers are with the Control Systems Technology Group, Department of Mechanical Engineering, Eindhoven University of Technology (e-mail: {a.rashidinejad, m.a.reniers}@tue.nl).

Martin Fabian is with Department of Electrical Engineering, Chalmers University of Technology, Sweden (e-mail: fabian@chalmers.se).

Supervisory control synthesis under communication delays was first investigated by Balemi [11]. To solve the problem, Balemi defines a condition called *delay insensitive language*. A plant has a delay insensitive language whenever any control command, enabled at a state of the plant, is not invalidated by an uncontrollable event. Under this condition, supervisory control under communication delays can be reduced to the conventional supervisory control synthesis [11]. In other words, if a given plant has a delay insensitive language, then the conventional supervisor is robust to the effects of delays. The benefit of this method is that nonblockingness and maximal permissiveness are already guaranteed by the supervisor if it exists (as they are guaranteed in the conventional supervisory control theory). However, the imposed condition restricts the applications for which such a supervisor exists.

In [12], [13], a condition called *delay-observability* is defined for the control requirement such that the existence of a networked supervisor depends on it. The delay-observability condition is similar to the delay insensitivity condition generalized for a sequence of uncontrollable events so that a control command is not invalidated by a sequence of consecutive uncontrollable events. In [12], [13], nonblockingness is guaranteed. However, maximal permissiveness is not guaranteed. Also, no method is proposed to obtain the supremal controllable and delay-observable sublanguage of a given control requirement [13].

In a more recent study, Lin introduced new observability and controllability conditions under the effects of communication delays called *network controllability* and *network observability* [14]. The approach presented by Lin has been further modified in [9], [14], [15], [19]–[22]. In all these works, the problem of supervisory control synthesis under communication delays is defined under certain conditions (network controllability and network observability or the modified versions of them). When the conditions are not met (by the control requirement), the synthesis does not result in a (networked) supervisor [9], [14], [19]–[21]. As discussed in [15], delayed observations and delayed control commands make it (more) challenging to ensure nonblockingness of the supervised plant (compared to the conventional non-networked setting when there is no delay). To guarantee nonblockingness, additional conditions are imposed on the control requirement in [9], but maximal permissiveness is not investigated.

In [16], an online predictive supervisory control synthesis method is presented to deal with control delays. The supervisor is claimed to be maximally permissive. However, this is not formally proved. This is also the case in [20] as they do not formally prove the maximal permissiveness although they establish the steps to achieve it. In [22], a predictive synthesis approach is proposed to achieve a networked supervisor which is guaranteed to be maximally permissive in case it satisfies the conditions. Nonblockingness is yet not investigated in [22]. None of the works following Lin's method consider simultaneously nonblockingness and maximal permissiveness. Moreover, as discussed in a recent study by Lin, in case that the conditions are not met by the control requirement, there is no method so far to compute the supremal sublanguage satisfying the conditions [23].

In [17], [18], a new synthesis algorithm is proposed in which the effects of communication delays are taken into account in the synthesis procedure instead of in extra conditions to be satisfied by the plant/control requirement. [17] investigates supervisory control of DES in an asynchronous setting. The asynchronous setting does not take time into account, but it is guaranteed that (if the algorithm terminates) the synthesized (asynchronous) supervisor satisfies nonblockingness. Maximal permissiveness is still an open issue in [17]. [18] focuses on timed delays, but it does not formally prove nonblockingness or maximal permissiveness.

In [24] as a more recent study, first, the control and observation channels are modeled. Then, both the plant and control requirements are transformed into a networked setting. Using these transformations, the problem of networked supervisory control synthesis is reduced to conventional supervisory control synthesis. Using conventional supervisory control synthesis, the resulting supervisor is controllable and nonblocking for the transformed plant and the transformed control requirements. However, it is not discussed if the supervisor satisfies these conditions for the (original) plant.

Furthermore, although it is important to consider time in the presence of delays, only a few papers investigate networked supervisory control of TDES [18], [19], [21], [25], [26] (where communication delays are modeled based on a consistent unit of time) as it introduces new complexities and challenges.

Table I gives an overview of the existing works. To the best of our knowledge, none of these works studies supervisory control synthesis of discrete-event systems under communication delays such that delays are modeled based on time, and the delivered supervisor guarantees both nonblockingness and maximal permissiveness as is done in this paper.

Citation	Timed	Nonblocking	Permissive
[11], [13]	✗	✓	✓
[14], [16], [20], [24]	✗	✗	✗
[9], [12], [17]	✗	✓	✗
[22]	✗	✗	✓
[18], [19], [21], [25], [26]	✓	✗	✗
This Paper	✓	✓	✓

TABLE I: Overview of existing works.

Our work is close to [18] in terms of the networked supervisory control setting and to [17] in terms of the synthesis technique. Similar to [18] and [17], the following practical conditions are taken into account:

1) A controllable event can be executed in the plant only if it is commanded (enabled) by the supervisor.

2) An uncontrollable event is not commanded (enabled) by the supervisor; it occurs spontaneously in the plant.

3) Any event, controllable or uncontrollable, executed in the plant is observable to the supervisor.

4) A control command sent by the supervisor reaches the plant after a constant amount of time delay. The command may not necessarily be accepted by the plant, in which case it will be removed from the control channel when the next *tick* occurs. Also, the observation of a plant event, controllable or uncontrollable, occurs after a constant amount of time delay.

5) The control channel is assumed to be FIFO, so control commands sent by the supervisor will reach the plant in the same order as they have been sent. However, the observation channel is non-FIFO, and so consecutive events that occur in the plant may be observed by the supervisor in any possible order. For instance, if the events a and b occur in that order between two *tick*s in the plant, they may be observed in the other order. Here, we investigate the situation where only the observation channel is non-FIFO. See Section III-C4 for a discussion on how the proposed solution is adapted for a non-FIFO control channel.

This paper improves [17], [18] in the following aspects:

1) Modeling purposes. In [5], a TDES is generally derived from a DES by restricting the execution of each event within a lower and an upper time bound specified to the event. Also, a TDES should satisfy the "activity-loop free" (ALF) assumption to guarantee that the clock never stops [5]. Fixing time bounds for events and imposing the ALF condition restrict the applications that can be modeled as TDESs. In this paper, the plant is already given as a TDES. Namely, the plant behavior is represented by an automaton, including the event *tick* with no specific relationship between the occurrences of *tick* and other events. To relax the ALF condition, the concept of time-lock freeness is introduced as a property, expressing the time progress of the system. Time-lock freeness, similar to nonblockingness, is guaranteed by the networked supervisor.

2) Synthesis technique. Inspired from the idea introduced in [17] to synthesize an asynchronous supervisor for DES, the synthesis method proposed in [18] for networked supervisory control of TDES is improved. For this purpose, first, the networked supervisory control (NSC) framework is modeled. Then, a networked plant automaton is proposed, modeling the behavior of the plant in the NSC framework. Based on the networked plant, a networked supervisor is synthesized. It is guaranteed that the networked supervisor provides nonblockingness, time-lock freeness, and maximal permissiveness.

3) Control requirement. The control requirement in [17], [18] is limited to the avoidance of illegal states. Here, the networked supervisory control synthesis is generalized to control requirements modeled as automata.

In the following, the NSC framework is introduced in Section II. For the NSC framework, an operator is proposed to give the networked supervised plant. Moreover, the conventional controllability and maximal permissiveness conditions are modified to timed networked controllability and timed networked maximal permissiveness conditions suitable for the NSC framework. Then, the basic networked supervisory control synthesis problem is formulated which aims to find a timed networked controllable and timed networked maximally permissive networked supervisor guaranteeing nonblockingness and time-lock freeness of the networked supervised plant. In Section III, first, the networked plant is defined as an automaton representing the behavior of the plant under communication delays and disordered observations. Furthermore, a technique is presented to synthesize a networked supervisor that is a solution to the basic networked supervisory control problem. In Section IV, the basic networked supervisory control synthesis problem is generalized to satisfy a given set of control requirements. Relevant examples are provided in each section. Finally, Section V concludes the paper. To enhance readability, all technical lemmas and proofs are given in the appendices.

II. Basic NSC Problem

A. Conventional Supervisory Control Synthesis of TDES

A TDES G is formally represented as a quintuple

$$G = (A, \Sigma, \delta, a_0, A_m),$$

where A, Σ, $\delta : A \times \Sigma \to A$, $a_0 \in A$, and $A_m \subseteq A$ stand for the set of states, the set of events, the (partial) transition function, the initial state, and the set of marked states, respectively. The set of events of any TDES is assumed to contain the event $tick \in \Sigma$. The set $\Sigma_a = \Sigma \setminus \{tick\}$ is called the set of active events. The notation $\delta(a, \sigma)!$ denotes that δ is defined for state a and event σ, i.e., there is a transition from state a with label σ to some state. The transition function is generalized to words in the usual way: $\delta(a, w) = a'$ means that there is a sequence of subsequent transitions from state a to the state a' that together make up the word $w \in \Sigma^*$. Starting from the initial state, the set of all possible words that may occur in G is called the language of G and is indicated by $L(G)$; $L(G) := \{w \in \Sigma^* \mid \delta(a_0, w)!\}$. Furthermore, for any state $a \in A$, the function $Reach(a)$ gives the set of states reachable from the state a; $Reach(a) := \{a' \in A \mid \exists w \in \Sigma^*, \delta(a, w) = a'\}$. States from which it is possible to reach a marked state are called nonblocking. An automaton is nonblocking when each state reachable from the initial state is nonblocking; for each $a \in Reach(a_0)$, $Reach(a) \cap A_m \neq \varnothing$. $L_m(G)$ denotes the marked language of G; $L_m(G) := \{w \in L(G) \mid \delta(a_0, w) \in A_m\}$. States from which time can progress are called *time-lock free* (TLF). An automaton is TLF when each state reachable from the initial state is TLF; for each $a \in Reach(a_0)$, there exists a $w \in \Sigma^*$ such that $\delta(a, w\,tick)!$.

Definition 1 (Natural Projection [4]): For sets of events Σ and $\Sigma' \subseteq \Sigma$, $P_{\Sigma'} : \Sigma^* \to \Sigma'^*$ is defined as follows: for $e \in \Sigma$ and $w \in \Sigma^*$,

$$P_{\Sigma'}(\varepsilon) := \varepsilon,$$

$$P_{\Sigma'}(we) := \begin{cases} P_{\Sigma'}(w)e & \text{if } e \in \Sigma', \\ P_{\Sigma'}(w) & \text{if } e \in \Sigma \setminus \Sigma'. \end{cases}$$

The definition of natural projection is extended to a language $L \subseteq \Sigma^*$; $P_{\Sigma'}(L) := \{w' \in \Sigma'^* \mid \exists w \in L, P_{\Sigma'}(w) = w'\}$ [4]. ■

Natural projection is an operation which is generally defined for languages. However, it is also possible to apply it on automata [27]. For an automaton with event set Σ, $P_{\Sigma'}$ first replaces all events not from Σ' by the silent event τ. Then, using a determinization algorithm (such as the one introduced in [28]), the resulting automaton is made deterministic. A state of a projected automaton is then marked if it contains at least one marked state from the original automaton (see [28] for more details). Using the notation δ_P for the transition function of the projected automaton, we state the following properties of this construction: (1) for any $w \in \Sigma^*$, if $\delta(a_0, w) = a_r$ then $\delta_P(A_0, P_{\Sigma'}(w)) = A_r$ where A_0 is the initial state of the projected automaton, and $A_r \subseteq A$ is a set with $a_r \in A_r$, (2) for any $w \in \Sigma^*$, if $\delta(a_0, w) \in A_m$, then $\delta_P(A_0, P_{\Sigma'}(w))$ is a marked state in the projected automaton.

In the rest of the paper, the plant is given as the TDES G represented by the automaton $(A, \Sigma_G, \delta_G, a_0, A_m)$ with $\Sigma_G = \Sigma_a \cup \{tick\}$ and $\Sigma_a \cap \{tick\} = \varnothing$. Also, as it holds for many applications, G is a finite automaton [5]. A finite automaton has a finite set of states and a finite set of events [29].

Here, it is assumed that all events in G are observable. A subset of the active events $\Sigma_{uc} \subseteq \Sigma_a$ is uncontrollable. $\Sigma_c = \Sigma_a \setminus \Sigma_{uc}$ gives the set of controllable active events. The event *tick* is uncontrollable by nature. However, as in [6], it is assumed that *tick* can be preempted by a set of forcible events $\Sigma_{for} \subseteq \Sigma_a$. Note that forcible events can be either controllable or uncontrollable. For instance, closing a valve to prevent overflow of a tank, and the landing of a plane are controllable and uncontrollable forcible events, respectively [5]. Note that for synthesis, the status of the event *tick* lies between controllable and uncontrollable depending on the presence of enabled forcible events. To clarify, when the event *tick* is enabled at some state a and also there exists a forcible event $\sigma \in \Sigma_{for}$ such that $\delta_G(a, \sigma)!$, then *tick* is considered as a controllable event since it can be preempted. Otherwise, *tick* is an uncontrollable event. In the figures, forcible events are underlined. The transitions labelled by controllable (active or *tick*) events are indicated by solid lines and the transitions labelled by uncontrollable (active or *tick*) events are indicated by dashed lines.

If the plant G is blocking, then a supervisor S needs to be synthesized to satisfy nonblockingness of the supervised plant. S is also a TDES with the same event set as G. Since the plant and supervisor are supposed to work synchronously in a conventional non-networked setting, the automaton representing the supervised plant behavior is obtained by applying the *synchronous product* indicated by $S||G$ [4]. Generally, in the synchronous product of two automata, a shared event can be executed only when it is enabled in both automata, and a non-shared event can be executed if it is enabled in the corresponding automaton. Since the conventional supervisor S has the same event set as G, each event will be executed in $S||G$ only if the supervisor enables (allows) it. S is controllable if it allows all uncontrollable events that may occur in the plant. This is captured in *conventional controllability for TDES.*

Definition 2 (Conventional Controllability for TDES (reformulated from [5])): Given a plant G with uncontrollable events Σ_{uc} and forcible events Σ_{for}, a TDES S, is controllable w.r.t. G if for all $w \in L(S||G)$ and $\sigma \in \Sigma_{uc} \cup \{tick\}$, if $w\sigma \in L(G)$,

1) $w\sigma \in L(S||G)$, or
2) $\sigma = tick$ and $w\sigma_f \in L(S||G)$ for some $\sigma_f \in \Sigma_{for}$. ■

Property (1) in the above definition is the standard controllability property (when there is no forcible event to preempt *tick*); S cannot disable uncontrollable events that G may generate. However, if a forcible event is enabled, this may preempt the time event, which is captured by Property (2).

A supervisor S is called *proper* for a plant G whenever S is controllable w.r.t. G, and the supervised plant $S||G$ is nonblocking.

Definition 3 (Conventional Maximal Permissivenesss): A proper supervisor S is *maximally permissive* for a plant G, whenever S preserves the largest behavior of G compared to any other proper supervisor S'; for any proper S': $L(S'||G) \subseteq L(S||G)$. ■

For a TDES, a proper and a maximally permissive supervisor can be synthesized by applying the synthesis algorithm proposed in [5].

B. Motivating Examples

This section discusses the situations where a proper and maximally permissive conventional supervisor S fails in the presence of observation delay (Example 1), non-FIFO observation (Example 2), or control delay (Example 3).

Example 1 (Observation Delay): Consider the plant depicted in Figure 1. To be maximally permissive, S must not disable a at a_0, and to be nonblocking, S must disable a at a_2. Now, assume that the observation of the events executed in G are not immediately received by S due to observation delay. Starting from a_0, imagine that u occurs, and G goes to a_2. Since S does not observe u immediately, it supposes that G is still at a_0 where it enables a. Then, a will be applied at the real state where G is, i.e., a_2, and so G goes to a_3 which is blocking.

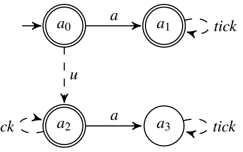

Fig. 1: Plant for Example 1.

Example 2 (Non-FIFO Observation): Consider the plant G depicted in Figure 2. To be nonblocking, S must disable a at a_3, and to be maximally permissive, S must not disable a at a_6. Now, assume that the observation channel is non-FIFO, i.e., events may be observed in a different order as they occurred in G. Starting from a_0, imagine that G executes $tick\,a\,b$ and goes to a_3. Since the observation channel is non-FIFO, S may receive the observation of $tick\,a\,b$ as $tick\,b\,a$ after which it does not disable a. However, G is actually at a_3 and by executing a, it goes to a_4 which is blocking.

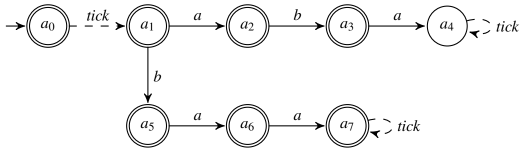

Fig. 2: Plant for Example 2.

Example 3 (Control Delay): Consider the plant depicted in Figure 3. To be maximally permissive, S must not disable a at a_1, and to be nonblocking S must disable a at a_3. Now, assume that control commands are received by G after one *tick*. Starting from a_0, S does not disable a after one *tick* (when G is at a_1). However, the command is received by G after the passage of one *tick* (due to the control delay) when G is at a_3. So, by executing a at a_3, G goes to a_4 which is blocking.

Remark 1: Conventional supervisory control synthesis of a TDES guarantees nonblockingness [5]. However, as can be

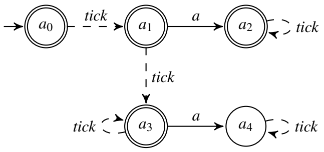

Fig. 3: Plant for Example 3.

seen in Example 2, it cannot guarantee time-lock freeness; a_3 is not TLF, and it is not removed by S. This is not an issue in [5] since a TDES is assumed to satisfy the ALF condition. Here, to guarantee time progress, the TLF property must be considered in synthesis.

As is clear from the examples, a supervisor is required that can deal with the problems caused by communication delays and disordered observations. To achieve such a supervisor, first, the networked supervisory control framework is established.

C. NSC Framework

In the presence of delays in the control and observation channels, enabling, executing and observing events do not happen at the same time. Figure 4 depicts the networked supervisory control (NSC) framework that is introduced in this paper. To recognize the differences between the enablement and observation of events and their execution in the plant, as in [17], [18], a set of *enabling events* Σ_e and a set of *observed events* Σ_o are introduced.

Definition 4 (Enabling and Observed Events): Given a plant G, to each controllable active event $\sigma \in \Sigma_c$ an enabling event $\sigma_e \in \Sigma_e$, and to each active event $\sigma \in \Sigma_a$ an observed event $\sigma_o \in \Sigma_o$ are associated such that $\Sigma_e \cap \Sigma_a = \varnothing$ and $\Sigma_o \cap \Sigma_a = \varnothing$ (clearly $\Sigma_e \cap \Sigma_o = \varnothing$). ■

Note that all events executed in the plant are supposed to be observable so that the observed event σ_o is associated to any $\sigma \in \Sigma_a$. However, not all the events are supposed to be controllable. Uncontrollable events such as disturbances or faults occur in the plant spontaneously. In this regard, enabling events σ_e are associated only to events from Σ_c.

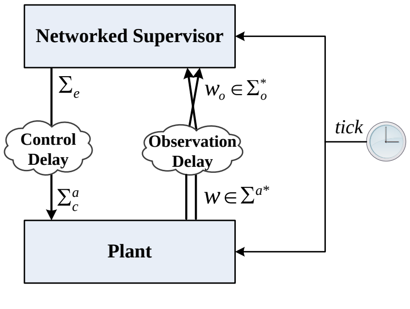

Fig. 4: NSC framework [18].

Considering Figure 4, a networked supervisor for G that fits in the proposed framework is a TDES given as:

$$NS = (Y, \Sigma_{NS}, \delta_{NS}, y_0, Y_m),$$

for which the event set $\Sigma_{NS} = \Sigma_e \cup \Sigma_o \cup \{tick\}$, and the event *tick* is produced by the global clock in the system so that $\Sigma_{NS} \cap \Sigma_G = \{tick\}$.

For the proposed NSC framework, the behavior of the plant under the control of a networked supervisor is achieved through *asynchronous composition*. To define asynchronous composition, we first need to consider the effects of delays on events sent through the control and observation channels. In this paper, it is assumed that the control (observation) channel has a finite capacity denoted by L_{max} (M_{max}), which introduces a constant amount of delay represented by a natural number N_c (N_o). Since the control channel is supposed to be FIFO, a list or sequence is used to consider the journey of events through the control channel. As given in Definition 5 below, $l \in (\Sigma_c \times [0, N_c])^*$ provides us with the current situation of the control channel. The interpretation of $l[i] = (\sigma, n)$ is that the i^{th} enabling event present in the control channel is σ_e which still requires n *ticks* before being received by the plant.

Definition 5 (Control Channel Representation): The control channel is represented by the set $L = (\Sigma_c \times [0, N_c])^*$. Moreover, we define the following operations for all $\sigma \in \Sigma_c$, the time counter $n \in [0, N_c]$ and $l \in L$:

- ε denotes the empty sequence.
- $app(l, (\sigma, n))$ adds the element (σ, n) to the end of l if $|l| < L_{max}$ (the channel is not full), otherwise l stays the same.
- $head(l)$ gives the first element of l (for nonempty lists). Formally, $head((\sigma, n)\ l) = (\sigma, n)$ and $head(\varepsilon)$ is undefined.
- $tail(l)$ denotes the list after removal of its leftmost element. Formally, $tail((\sigma, n)\ l) = l$ and $tail(\varepsilon)$ is undefined.
- $l - 1$ decreases the natural number component of every element in l by one (if possible). It is defined inductively as follows $\varepsilon - 1 = \varepsilon$, $((\sigma, 0)\ l) - 1 = \ l - 1$, and $((\sigma, n + 1)\ l) - 1 = (\sigma, n)\ (l - 1)$. ■

Due to the assumption that the observation channel is non-FIFO, we use a multiset to consider the journey of each event through the observation channel. As given in Definition 6 below, the multiset $m : \Sigma_a \times [0, N_o] \to \mathbb{N}$ provides us with the current situation of the observation channel. The interpretation of $m(\sigma, n) = k$ is that currently there are k events σ in the observation channel that still require n *ticks* before reaching the (networked) supervisor.

Definition 6 (Observation Channel Representation): The observation channel is represented by the set $M = \{m \,|\, m : \Sigma_a \times [0, N_o] \to \mathbb{N}\}$. Moreover, we define the following operations for all $m \in M$, $\sigma, \sigma' \in \Sigma_a$ and the time counters $n, n' \in [0, N_o]$:

- $[]$ denotes the empty multiset, i.e., the function m with $m(\sigma, n) = 0$.
- $|m| = \sum_{(\sigma,n) \in \Sigma_a \times [0,N_o]} m(\sigma, n)$ denotes the number of events in the observation channel represented by m.
- $m \uplus [(\sigma, n)]$ inserts (σ, n) to m if $|m| < M_{max}$ (the observation channel is not full). Formally, it denotes the function m' for which $m'(\sigma, n) = m(\sigma, n) + 1$ and $m'(\sigma', n') = m(\sigma', n')$ otherwise. If $|m| = M_{max}$ (the observation channel is full), then the channel stays the same, i.e., $m' = m$.

- $m \setminus [(\sigma,n)]$ removes (σ,n) from m once. Formally, it denotes the function m' for which $m'(\sigma,n) = \max(m(\sigma,n)-1,0)$ and $m'(\sigma',n') = m(\sigma',n')$ otherwise.
- $m-1$ decreases the natural number component of every element by one (as long as it is positive). Formally, it denotes the function m' for which $m'(\sigma,n) = m(\sigma,n+1)$ for all $n < N_o$ and $m'(\sigma,N_o) = 0$.
- $(\sigma,n) \in m$ denotes that the pair (σ,n) is present in m, it holds if $m(\sigma,n) > 0$. ■

In the rest of the paper, a networked supervisor for the plant G is given as the TDES *NS* represented by the automaton $(Y, \Sigma_{NS}, \delta_{NS}, y_0, Y_m)$.

Considering the representation of control and observation channels, an asynchronous composition operator is defined to achieve a networked supervised plant.

Definition 7 (Timed Asynchronous Composition Operator): Given a plant G and a networked supervisor *NS* (for G), the asynchronous product of G and *NS*, denoted by $NS_{N_c}\|_{N_o} G$, is given by the automaton

$$NS_{N_c}\|_{N_o} G = (Z, \Sigma_{NSP}, \delta_{NSP}, z_0, Z_m),$$

where

$$Z = A \times Y \times M \times L, \qquad \Sigma_{NSP} = \Sigma_{NS} \cup \Sigma,$$
$$z_0 = (a_0, y_0, [], \varepsilon), \qquad Z_m = A_m \times Y_m \times M \times L.$$

Moreover, for $a \in A$, $y \in Y$, $m \in M$, and $l \in L$, $\delta_{NSP} : Z \times \Sigma_{NSP} \to Z$ is defined as follows:

1) When an event $\sigma_e \in \Sigma_e$ occurs in *NS*, it is sent through the control channel. This is represented by adding (σ, N_c) to l where N_c is the remaining time for σ_e until being received by G. If $\delta_{NS}(y,\sigma_e)!$:
$$\delta_{NSP}((a,y,m,l),\sigma_e) = (a, \delta_{NS}(y,\sigma_e), m, app(l,(\sigma,N_c))).$$
2) An active controllable event $\sigma \in \Sigma_c$ can occur if the plant enables it, and the corresponding control command (enabling event) is received by the plant as $(\sigma,0)$ (as the enabling event finished its journey through the control channel). When σ occurs, it will be stored in m with the remaining time N_o until being observed by *NS*. If $\delta_G(a,\sigma)!$ and $head(l) = (\sigma,0)$:
$$\delta_{NSP}((a,y,m,l),\sigma) = (\delta_G(a,\sigma), y, m \uplus [(\sigma,N_o)], tail(l)).$$
3) An uncontrollable event $\sigma \in \Sigma_{uc}$ can occur if it is enabled in G. When σ occurs, it will be stored in m with the remaining time N_o until being observed by *NS*. If $\delta_G(a,\sigma)!$:
$$\delta_{NSP}((a,y,m,l),\sigma) = (\delta_G(a,\sigma), y, m \uplus [(\sigma,N_o)], l).$$
4) Event *tick* can occur if both *NS* and G enable it, and there is no event ready to be observed by *NS*. Upon the execution of *tick*, all the time counters in m and l are decreased by one. If $\delta_G(a,tick)!$, $\delta_{NS}(y,tick)!$, $(\sigma,0) \notin m$ for all $\sigma \in \Sigma_a$
$$\delta_{NSP}((a,y,m,l),tick) = (\delta_G(a,tick), \delta_{NS}(y,tick), m-1, l-1).$$
5) The observation of an active event $\sigma \in \Sigma_a$ can occur when it finishes its journey through the observation channel (and so it is received by *NS*), and σ_o is enabled by *NS*. When σ_o occurs, $(\sigma,0)$ is removed from m. If $\delta_{NS}(y,\sigma_o)!$ and $(\sigma,0) \in m$:
$$\delta_{NSP}((a,y,m,l),\sigma_o) = (a, \delta_{NS}(y,\sigma_o), m \setminus [(\sigma,0)], l). \quad ■$$

In the rest of the paper, the asynchronous composition $NS_{N_c}\|_{N_o} G$ of the plant G and the networked supervisor *NS* (for that plant) is assumed to be the TDES *NSP* represented by the automaton $(Z, \Sigma_{NSP}, \delta_{NSP}, z_0, Z_m)$.

Note that the networked supervised plant models the behavior of a plant controlled by a networked supervisor, and so for the proposed operator, we need to prove that the result does not enlarge the behavior of the plant.

Property 1 (NSP and Plant): Given a plant G and networked supervisor *NS* (for that plant): $P_{\Sigma_G}(L(NSP)) \subseteq L(G)$.

Proof: See Appendix B-A. ■

A networked supervisor is controllable with respect to a plant if it never disables any uncontrollable event that can be executed by the plant. To have a formal representation of controllability in the NSC framework, Definition 2 is adapted to *timed networked controllability*.

Definition 8 (Timed Networked Controllability): Given a plant G with uncontrollable events Σ_{uc} and forcible events Σ_{for}, a networked supervisor *NS*, is controllable w.r.t. G if for all $w \in L(NSP)$ and $\sigma \in \Sigma_{uc} \cup \{tick\}$, whenever $P_{\Sigma_G}(w)\sigma \in L(G)$:

1) $w\sigma \in L(NSP)$, or
2) $\sigma = tick$ and $w\sigma_f \in L(NSP)$ for some $\sigma_f \in \hat{\Sigma}_{for} \cup \Sigma_o$, where $\hat{\Sigma}_{for} = \Sigma_{for} \cup \Sigma_e$. ■

When there is no network, i.e., $\Sigma_{NS} = \Sigma_G$, timed networked controllability coincides with conventional controllability for TDES (Definition 2).

Remark 2: Considering Definition 7, *tick* does not occur if there is an event ready to be observed ($(\sigma,0) \in m$). In other words, observed events always preempt *tick* since they occur once they finish their journey in the observation channel. The enabling events are assumed to be forcible as well. This gives the opportunity to the networked supervisor to preempt *tick* by enabling an event whenever it is necessary. In Section III-C, we discuss other possible cases.

A networked supervisor *NS* is called proper in NSC framework if is timed networked controllable, nonblocking, and TLF. Similar to controllability, the definition of maximal permissiveness (in the conventional setting) is adapted to *timed networked maximal permissiveness* (for NSC Framework).

Definition 9 (Timed Networked Maximal Permissiveness): A proper networked supervisor *NS* is timed networked maximally permissive for a plant G, if for any other proper networked supervisor NS' in the same NSC framework (with event set Σ_{NS}): $P_{\Sigma_G}(L(NS'_{N_c}\|_{N_o} G)) \subseteq P_{\Sigma_G}(L(NSP)$. In other words, *NS* preserves the largest admissible behavior of G. ■ Again, when there is no network, this notion coincides with conventional maximal permissiveness (Definition 3).

D. Problem Formulation

The *Basic NSC Problem* is defined as follows. Given a plant model G as a TDES, observation (control) channel with delay N_o (N_c) and maximum capacity M_{max} (L_{max}), provide a networked supervisor NS such that

- NSP is nonblocking,
- NSP is time-lock free
- NS is timed networked controllable for G, and
- NS is timed networked maximally permissive.

III. Networked Supervisory Control Synthesis

To achieve a proper and maximally permissive networked supervisor (in the NSC framework), the synthesis is applied on the "networked plant", as indicated in Figure 5. The networked plant is a model for how events are executed in the plant according to the enabling events, and how the observations of the executed events may occur in a networked supervisory control setting. Based on the networked plant, a synthesis algorithm is proposed to obtain a networked supervisor, which is a solution to the basic NSC problem. Example 4 is used to illustrate each step of the approach.

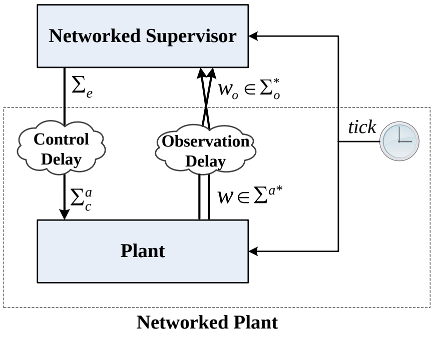

Fig. 5: Networked plant.

Example 4: (Endangered Pedestrian) Let us consider the endangered pedestrian example from [5]. The plant G is depicted in Figure 6. Both the bus and pedestrian are supposed to do single transitions indicated by p for passing and j for jumping. The requirement considered in [5] is that the pedestrian should jump before the bus passes. However, since we do not consider requirements here (yet), we adapt the plant from [5] such that if the bus passes before the pedestrian jumps, then G goes to a blocking state. The control channel is FIFO, the observation channel is non-FIFO, $N_c = N_o = 1$, $L_{max} = 1$, and $M_{max} = 2$. We aim to synthesize a proper and maximally permissive networked supervisor for G.

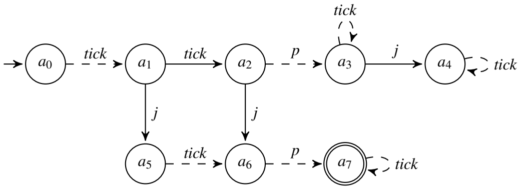

Fig. 6: Endangered pedestrian from Example 4.

A. Networked Plant

The behavior of the plant communicating through the control and observation channels is captured by the *networked plant*. As is clear from Figure 5, if we do not consider enabling and observation of events, what is executed in the networked plant is always a part of the plant behavior. Let us denote by NP the networked plant automaton, then $P_{\Sigma_G}(L(NP)) \subseteq L(G)$.

Moreover, note that a networked supervisor is synthesized for a plant on the basis of the networked plant. The networked plant should represent all the possible behavior of the plant in the networked supervisory control setting, and it is only the networked supervisor that may prevent the occurrence of some plant events by disabling the relevant enabling event. This means that NP should be such that $L(G) \subseteq P_{\Sigma_G}(L(NP))$. The latter property relies on the following assumptions.

Assumption 1: The plant enables enough *ticks* in the beginning; there are at least N_c *ticks* (there can be uncontrollable events occurring between *ticks*) enabled before the first controllable event.

Assumption 2: The control channel provides enough capacity for all enabling commands being sent to the plant. Imagine that $tick\,\sigma\,tick^* \in L(G)$, and $L_{max} = 0$. Then, σ_e may occur in NP, but the plant will never execute σ as it does not receive the relevant enabling command. To avoid this situation, the size of the control channel should be such that it always has the capacity for all enabling events. An enabling event will be removed from the control channel after N_c *ticks*. So, considering all substrings w that can appear in the plant (after an initial part w_0) which are no longer (in the time sense) than N_c *ticks*, then the control channel capacity should be at least equal to the number of controllable events occurring in w; $L_{max} \geq \max_{w \in W} |P_{\Sigma_c}(w)|$ where $W = \{w \in \Sigma_G^* \mid \exists w_0 w \in L(G), |P_{\{tick\}}(w)| \leq N_c - 1\}$.

To obtain the networked plant, we present the function Π in Definition 10. In order to determine enabling commands we look N_c *ticks* ahead for only the controllable active events enabled in $G' = P_{\Sigma_G \setminus \Sigma_u}(G)$. We use a list L to store the controllable events that have been commanded and a medium M to store the events that were executed.

Definition 10 (Networked Plant Operator): For a given plant, G, Π gives the networked plant as:

$$\Pi(G, N_c, N_o, L_{max}, M_{max}) = (X, \Sigma_{NSP}, \delta_{NP}, x_0, X_m),$$

Let $G' = P_{\Sigma_G \setminus \Sigma_u}(G) = (A', \Sigma_G, \delta'_G, a'_0, A'_m)$, and

$$X = A \times A' \times M \times L, \qquad x_0 = (a_0, \delta'_G(a'_0, tick^{N_c}), [], \varepsilon),$$
$$X_m = A_m \times A' \times M \times L.$$

For $a \in A$, $a' \in A'$, $m \in M$ and $l \in L$, the transition function $\delta_{NP} : X \times \Sigma_{NSP} \to X$ is defined as follows:

1) If $\delta'_G(a', \sigma)!$, $\sigma \in \Sigma_c$

$$\delta_{NP}((a, a', m, l), \sigma_e) = (a, \delta'_G(a', \sigma), m, app(l, (\sigma, N_c))).$$

2) If $\delta_G(a, \sigma)!$, $head(l) = (\sigma, 0), \sigma \in \Sigma_c$

$$\delta_{NP}((a, a', m, l), \sigma) = (\delta_G(a, \sigma), a', m \uplus [(\sigma, N_o)], tail(l)).$$

3) If $\delta_G(a, \sigma)!, \sigma \in \Sigma_{uc}$

$$\delta_{NP}((a, a', m, l), \sigma) = (\delta_G(a, \sigma), a', m \uplus [(\sigma, N_o)], l).$$

4) If $\delta_G(a,tick)!$, $\neg\delta'_G(a',\sigma)!$ for all $\sigma\in\Sigma_c$, and $(\sigma',0)\notin m$ for all $\sigma,\sigma'\in\Sigma_a$

$$\delta_{NP}((a,a',m,l),tick)=$$
$$\begin{cases}(\delta_G(a,tick),\delta'_G(a',tick),m-1,l-1) & \text{if } \delta'_G(a',tick)!,\\ (\delta_G(a,tick),a',m-1,l-1) & \text{otherwise.}\end{cases}$$

5) If $(\sigma,0)\in m$

$$\delta_{NP}((a,a',m,l),\sigma_o)=(a,a',m\setminus[(\sigma,0)],l). \qquad \blacksquare$$

Note that due to Assumption 1, $\delta'_G(a'_0,tick^{N_c})$ is always defined. In the rest of the paper, the networked plant of the plant G is assumed to be the TDES NP represented by the automaton $(X,\Sigma_{NSP},\delta_{NP},x_0,X_m)$.

Property 2 (NP and Plant): For any plant G:

1) $P_{\Sigma_G}(L(NP))\subseteq L(G)$, and
2) $L(G)\subseteq P_{\Sigma_G}(L(NP))$ whenever assumptions 1 and 2 hold.

Proof: See Appendix B-B. ■

Example 5: For the endangered pedestrian from Example 4, G' and NP are given in Figure 7 and Figure 8, respectively.

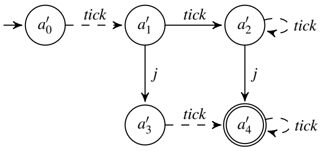

Fig. 7: G' for the endangered pedestrian from Example 4.

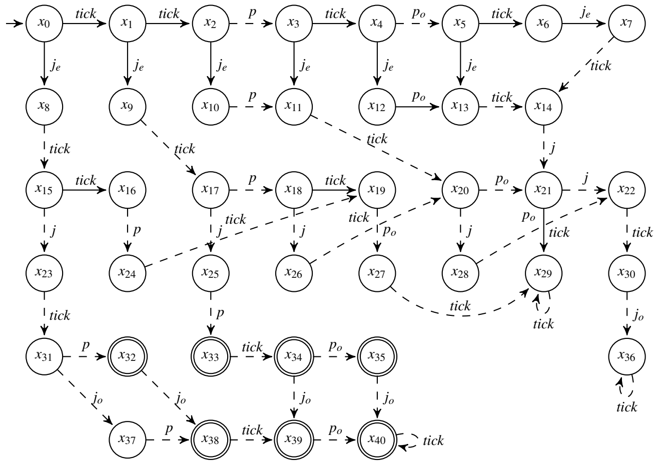

Fig. 8: Networked plant for the endangered pedestrian from Example 4 ($N_c=1,N_o=1$).

B. Synthesis

As is clear from Figure 5, enabling events are the only controllable events that can be disabled by the networked supervisor. All other events in the networked plant (active events and observed events) are uncontrollable. Moreover, controllability of *tick* depends on the forcible events of the plant as well as the enabling events (as we assume that they are forcible). To clarify, uncontrollable events are indicated by dashed lines in Figure 8. Note also that the observed events are observable to the networked supervisor. Also, events from Σ_e are observable, as the networked supervisor knows about the commands that it sends to the plant. However, the events from Σ_a are now unobservable to the networked supervisor. To consider these issues in the current step of the approach, the sets of unobservable events $\hat{\Sigma}_{uo}$, observable events $\hat{\Sigma}_o$, uncontrollable active events $\hat{\Sigma}_{uc}$, and controllable active events $\hat{\Sigma}_c$ of the networked plant are given by $\hat{\Sigma}_{uo}=\Sigma_a$, $\hat{\Sigma}_o=\Sigma_e\cup\Sigma_o\cup\{tick\}$, $\hat{\Sigma}_{uc}=\Sigma_a\cup\Sigma_o$, $\hat{\Sigma}_c=\Sigma_e$. Also, as mentioned before $\hat{\Sigma}_{for}=\Sigma_{for}\cup\Sigma_e$. The event *tick* is always observable to the networked supervisor. Moreover, it is uncontrollable unless there exists an event from $\hat{\Sigma}_{for}$ enabled in parallel to *tick*. Regarding the new sets of events, the synthesis algorithm takes into account the TDES conventional controllability (in Definition 2) and is inspired from the weak observability condition introduced in [30], [31].

Algorithm 1 presents the synthesis procedure in which we use the following additional concepts and abbreviations:

- $BS(NS)=BLock(NS)\cup TLock(NS)$ where $BLock(NS)$ gives the set of blocking states of NS, and $TLock(NS)$ gives the set of time-lock states of NS.
- Due to the fact that events from Σ_a are unobservable in the networked plant, one should be careful that the same control command is applied on the states reachable through the same observations. To take this issue into account, the following function is used in the synthesis algorithm; $OBS(x)=\{x'\in X\mid \exists w,w'\in\Sigma^*_{NP},\delta_{NP}(x_0,w)=x\wedge\delta_{NP}(x_0,w')=x'\wedge P_{\hat{\Sigma}_o}(w)=P_{\hat{\Sigma}_o}(w')\}$ gives the set of states observationally equivalently reachable as x. The function OBS can be applied on a set of states $X'\subseteq X$ as well such that $OBS(X')=\bigcup_{x\in X'}OBS(x)$.
- $F(y)=\{\sigma\in\hat{\Sigma}_{for}\mid\delta_{NS}(y,\sigma)!\}$ is the set of forcible events enabled at state y.
- Besides blocking and time-lock states, we should take care of states from which a state from $BS(NS)$ can be reached in an uncontrollable way, taking preemption of *tick* events into account. $Uncon(BS(NS))$ gives a set of states, called *bad states*, such that
    1) $BS\subseteq Uncon(BS(NS))$;
    2) if $\delta_{NS}(y,\sigma)\in Uncon(BS(NS))$ for some $y\in Y$ and $\sigma\in\hat{\Sigma}_{uc}$, then $y\in Uncon(BS(NS))$;
    3) if $\delta_{NS}(y,tick)\in Uncon(BS(NS))$ for some $y\in Y$ such that for all $y'\in OBS(y)$, $F(y)\cap F(y')=\varnothing$, then $y\in Uncon(BS(NS))$. This is to make sure that the supervisor behaves the same towards all observationally equivalent transitions.
- $BPre(NS)=\{y\in Y\mid F(y)=0\ \wedge\ \neg\delta_{NS}(y,tick)!\ \wedge\ \delta_{NP}(y,tick)!\}$ contains states (still in NS) from which no forcible events and no *tick* are enabled while there was a *tick* event enabled in the networked plant.
- $Reach(NS)$ restricts an automaton to those states that are reachable from the initial state.

Starting from $NS=NP$, Algorithm 1 changes NS by disabling transitions at line 8 and delivering the reachable part at line 11. For the proposed algorithm, the following property and theorems hold.

Algorithm 1 Networked supervisory control synthesis
Input: $NP = (X,\Sigma_{NSP},\delta_{NP},x_0,X_m)$, $\hat{\Sigma}_{uo}$, $\hat{\Sigma}_{uc}$, $\hat{\Sigma}_c$, $\hat{\Sigma}_{for}$
Output: $NS = (Y,\Sigma_{NS},\delta_{NS},y_0,Y_m)$

```
$i \leftarrow 0$
$ns(0) \leftarrow NP$
$bs(0) \leftarrow BS(ns(0))$
while $y_0 \notin Uncon(bs(i)) \wedge bs(i) \neq \varnothing$ do
    for $y \in Y \setminus Uncon(bs(i))$ and $\sigma \in \hat{\Sigma}_c \cup \{tick\}$ do
        if $\delta_{NS}(y,\sigma) \in OBS(Uncon(bs(i)))$ then
            for $y' \in OBS(y)$ do
                $\delta_{NS}(y',\sigma) \leftarrow$ undefined
    $Y \leftarrow Y \setminus Uncon(bs(i))$
    $i \leftarrow i+1$
    $ns(i) \leftarrow Reach(ns(i-1))$
    $bs(i) \leftarrow BPre(ns(i)) \cup BS(ns(i))$
if $y_0 \in Uncon(bs(i))$ then
    no result
$NS \leftarrow P_{\Sigma_{NSP}\setminus\Sigma}(ns(i))$
```

Property 3 (Algorithm Termination): The synthesis algorithm presented in Algorithm 1 terminates.

Proof: See Appendix B-C. ■

Theorem 1 (Nonblocking NSP): Given a plant G and the networked supervisor NS computed by Algorithm 1: NSP is nonblocking.

Proof: See Appendix B-D. ■

Theorem 2 (TLF NSP): Given a plant G and the networked supervisor NS computed by Algorithm 1: NSP is TLF.

Proof: See Appendix B-E. ■

Theorem 3 (Controllable NS): Given a plant G and the networked supervisor NS computed by Algorithm 1: NS is timed networked controllable w.r.t. G.

Proof: See Appendix B-F. ■

Theorem 4 (Timed Networked Maximally Permissive NS): For a plant G, the networked supervisor NS computed by Algorithm 1 is timed networked maximally permissive.

Proof: See Appendix B-G. ■

C. Possible Variants

The proposed synthesis approach can be adjusted for the following situations.

1) Nonblockingness or time-lock freeness: Algorithm 1 can easily be adapted to either only provide nonblockingness or time-lock freeness by removing $TLock(NS)$ and $BLock(NS)$ from $BS(NS)$, respectively.

2) Unobservable enabling events: We could have assumed that some events from Σ_e are unobservable. In this case, $\Sigma_a \subseteq \hat{\Sigma}_{uo} \subseteq \Sigma_a \cup \Sigma_e$, and so there would be more states that become observationally equivalent. Hence, the resulting supervisor could be more restrictive since a control command should be disabled at all observationally equivalent states if it needs to be disabled at one of them. Also if the observation channel does not provide enough capacity, more states become observationally equivalent, resulting in a more conservative solution. To not introduce any observation losses, the observation channel needs to be such that it has the capacity for all observations of events executed in the plant; $M_{max} \geq \max_{w\in W}\{|P_{\Sigma_a}(w)|\}$ where $W = \{w \in \Sigma_G^* \mid \exists w_0 w \in L(G), |P_{\{tick\}}(w)| \leq N_o\}$ as all events are observed after N_o *ticks*.

3) Non-forcible enabling events: We could have assumed that some events from Σ_e are not forcible. In this case, $\Sigma_{for} \subseteq \hat{\Sigma}_{for} \subseteq \Sigma_{for} \cup \Sigma_e$. Providing less forcible events makes the synthesis result more conservative since if the non-preemptable *tick* leads to a bad state, the current state where *tick* is enabled must be avoided as well (illustrated by Example 6).

Example 6: Consider the endangered pedestrian from Example 5. With the assumption that events from Σ_e are forcible, the networked supervisor is given in Figure 9. Without this assumption, there exists no networked supervisor.

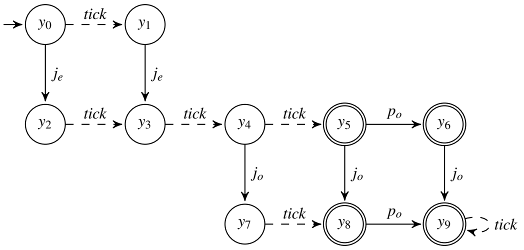

Fig. 9: Networked supervisor for the endangered pedestrian from Example 4 ($N_c = 1, N_o = 1$).

4) Non-FIFO control channel: Our proposed framework can easily be extended to the case that the control channel is non-FIFO by applying the following changes. Similar to the observation channel, the control channel is represented by $L = \{l \mid l : \Sigma \times [0,N_c] \to \mathbb{N}\}$ where l is a multiset. So, for each $l \in L$ and the time counter n, we define the operators $l \uplus [(\sigma,n)]$ and $l \setminus [(\sigma,0)]$ instead of $app(l,(\sigma,n))$ and $tail(l)$, respectively. This affects item 1) of both Definition 7 and Definition 10 such that (σ,N_c) is simply added to l without taking into account the order of elements. Also, in item 2) of both definitions, $head(l)$ is replaced by $\exists(\sigma,0) \in l$. This may change the result pretty much as the enabling events can now be received by G in any possible order. As Example 7 illustrates, this may increase the chance of reaching blocking or time-lock states and result in very conservative solutions for many applications.

Example 7: Given a plant G indicated in Figure 10, $N_c = N_o = 1$, and $L_{max} = M_{max} = 1$, NP is obtained as in Figure 11. The networked supervisor computed by Algorithm 1 only disables the event b_e at x_0. Now, assume that the control channel is non-FIFO as well. Then, at $x_3 = (\delta_G(a_0,tick),\delta'(a'_0,tick\,a\,b\,tick),[],(a,0)(b,0))$, b can be executed as well as a. By executing b at x_3, NP goes to a blocking state. In this case, Algorithm 1 returns no result since x_0 becomes a blocking state and needs to be removed.

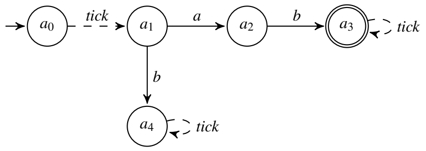

Fig. 10: Plant from Example 7.

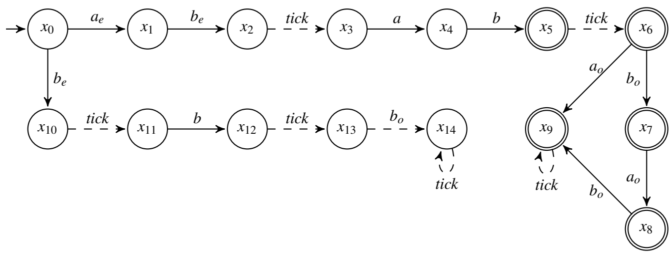

Fig. 11: Networked plant from Example 7.

IV. Requirement Automata

To generalize the method to a wider group of applications, we solve the basic NSC problem for a given set of control requirements. It is assumed that the desired behavior of G, denoted by the TDES R, is represented by the automaton $(Q,\Sigma_R,\delta_R,q_0,Q_M)$ where $\Sigma_R \subseteq \Sigma_G$. Since most control requirements are defined to provide safety of a plant, we call a supervised plant *safe* if it satisfies the control requirements.

Definition 11 (Safety): Given a plant G and requirement R, a TDES NSP with event set Σ_{NSP} is safe w.r.t. G and R if its behavior stays within the legal/safe behavior as specified by R; $P_{\Sigma_{NSP}\cap\Sigma_R}(L(NSP)) \subseteq P_{\Sigma_{NSP}\cap\Sigma_R}(L(R))$. ■

Problem Statement: Given a plant model G as a TDES, control requirement R for G (also a TDES), observation (control) channel with delay N_o (N_c) and maximum capacity M_{max} (L_{max}), provide a networked supervisor NS such that

- NSP is nonblocking,
- NSP is time-lock free,
- NS is timed networked controllable w.r.t. G,
- NS is timed networked maximally permissive, and
- NSP is safe for G w.r.t. R.

In the conventional non-networked supervisory control setting, if R is controllable w.r.t. G (as defined in Definition 2), then an optimal nonblocking supervisor can be synthesized for G satisfying R [5]. If R is not controllable w.r.t. G, then the supremal controllable sublanguage of $G||R$, indicated by $sup\mathscr{C}(G||R)$, should be calculated. Then, the synthesis is applied on $sup\mathscr{C}(G||R)$ [4], [5].

In a networked supervisory control setting, synthesizing a networked supervisor for $sup\mathscr{C}(G||R)$ does not always result in a safe networked supervised plant. This issue occurs due to the fact that in $sup\mathscr{C}(G||R)$, some events are already supposed to be disabled, to deal with controllability problems introduced by requirement R. In a conventional non-networked setting, this does not cause a problem because events are observed immediately when executed. However, when observations are delayed, there could be a set of states reached by the same observation. Hence, if an event is disabled at a state, it should be disabled at all observationally equivalent ones. Even for a controllable requirement, any disablement of events should be considered at all observationally equivalent states.

To take care of this issue, any requirement automaton R (whether controllable or uncontrollable) is made complete as $R^{\perp}$ in terms of both uncontrollable and controllable events. *Completion* was first introduced in [32] where the requirement automaton R is made complete in terms of only uncontrollable events. By applying the synthesis on $G||R^{\perp}$, all original controllability problems in $G||R$ are translated to blocking issues. Note that this translation is necessary to let the supervisor know about the uncontrollable events that are disabled by a given requirement. To solve the blocking issues, synthesis still takes the controllability definition into account.

Definition 12 (Automata Completion): For a TDES $R = (Q,\Sigma_R,\delta_R,q_0,Q_M)$, the complete automaton $R^{\perp}$ is defined as $R^{\perp} = (Q\cup\{q_d\},\Sigma_R,\delta_R^{\perp},q_0,Q_M)$ with $q_d \notin Q$, where for every $q \in Q$ and $\sigma \in \Sigma_R$,

$$\delta_R^{\perp}(q,\sigma) = \begin{cases} \delta_R(q,\sigma) & \text{if } \delta_R(q,\sigma)! \\ q_d & \text{otherwise.} \end{cases}$$

■

To find a networked supervisor, Algorithm 1 is applied on $\Pi(G||R^{\perp},N_c,N_o,L_{max},M_{max})$. The obtained networked supervisor is already guaranteed to be timed networked controllable, timed networked maximally permissive, and it results in a nonblocking and time-lock free networked supervised plant. Theorem 5 shows that the networked supervised plant is safe as well.

Theorem 5 (Safe NSP): Given a plant G, requirement R, and the networked supervisor NS computed by Algorithm 1 for $\Pi(G||R^{\perp},N_c,N_o,L_{max},M_{max})$: $NS_{N_c}||_{N_o}(G||R^{\perp})$ is safe for G w.r.t. R.

Proof: See Appendix B-H. ■

V. Conclusions and Future Work

In this paper, we study the networked supervisory control synthesis problem. We first introduce a networked supervisory control framework in which both control and observation channels introduce delays, the control channel is FIFO, and the observation channel is non-FIFO. Moreover, we assume that a global clock exists in the system such that the passage of a unit of time is considered as an event *tick* in the plant model. Also, communication delays are measured as a number of occurrences of the *tick* event. In our framework, uncontrollable events occur in the plant spontaneously. However, controllable events can be executed only if they have been enabled by the networked supervisor. On the other hand, the plant can either accept a control command (enabled by the networked supervisor) and execute it or ignore the control command and execute some other event. For the proposed framework, we also provide an asynchronous composition operator to obtain the networked supervised plant. Furthermore, we adapt the definition of conventional controllability for our framework and introduce timed networked controllability. Then, we present a method of achieving the networked plant automaton representing the behavior of the plant in the networked supervisory control framework. For the networked plant, we provide an algorithm synthesizing a networked supervisor which is timed networked controllable, nonblocking, time-lock free, and maximally permissive. Finally, to generalize, we solve the problem for a given set of control (safety) requirements modeled as automata. We guarantee that the proposed technique achieves a networked supervisor that is timed networked controllable, nonblocking, time-lock free, maximally permissive, and safe.

Our proposed approach can be adjusted to a setting with observation delay and control delay specified to each event, a setting with bounded control and observation delays, or to a setting with lossy communication channels. In each case, only the timed asynchronous composition and networked plant operators need to be updated, the synthesis algorithm stays the same. For cases with large state spaces, we must deal with the scalability problem of the networked plant. For such cases, it is suggested to switch to timed automata. A supervisory control synthesis method for timed automata has been recently proposed by the authors [33]. Networked supervisory control of timed automata will be investigated in future research.

Appendix A
Technical Lemmas

Here, the notation . is used to refer to an element of a tuple. For instance, $z.a$ refers to the (first) element a of $z = (a, y, m, l)$.

Lemma 1 (Nonblockingness over Projection [18]): For any TDES G with event set Σ and any event set $\Sigma' \subseteq \Sigma$: if G is nonblocking, then $P_{\Sigma'}(G)$ is nonblocking.

Proof: Consider an arbitrary TDES $G = (A, \Sigma, \delta, a_0, A_m)$ and arbitrary $\Sigma' \subseteq \Sigma$. Suppose that G is nonblocking. Consider an arbitrary reachable state $A_r \subseteq A$ in $P_{\Sigma'}(G)$. By construction A_r is nonempty. Assume that this state is reached through the word $w \in \Sigma'$. Then, for each state $a \in A_r$, again by construction, $\delta(a_0, w') = a$ for some $w' \in \Sigma^*$ with $P_{\Sigma'}(w') = w$. Because G is nonblocking, there exists a $v' \in \Sigma^*$ such that $\delta(a, v') = a_m$ for some $a_m \in A_m$. Consequently, from state A_r, it is possible to have a transition labelled with $P_{\Sigma'}(v')$ to a state A'_r containing a_m. By construction, this state A'_r is a marked state in $P_{\Sigma'}(G)$. Hence, the projection automaton is nonblocking as well. ■

Lemma 2 (Time-lock Freeness over Projection [18]): For any TDES G with event set Σ and any event set $\Sigma' \subseteq \Sigma$, $tick \in \Sigma'$: if G is TLF , then $P_{\Sigma'}(G)$ is TLF.

Proof: The proof is similar to the proof of Lemma 1. ■

Lemma 3 (NSP Transitions): Given a plant G, networked supervisor NS (for that plant) and networked supervised plant NSP (for those): $\delta_{NSP}(z_0, w).a = \delta_G(a_0, P_{\Sigma_G}(w))$ and $\delta_{NSP}(z_0, w).y = \delta_{NS}(y_0, P_{\Sigma_{NS}}(w))$, for any $w \in L(NSP)$.

Proof: Take $w \in L(NSP)$, we show that $\delta_{NSP}(z_0, w).a = \delta_G(a_0, P_{\Sigma_G}(w))$ and $\delta_{NSP}(z_0, w).y = \delta_{NS}(y_0, P_{\Sigma_{NS}}(w))$. This is proved by induction on the structure of w. **Base case:** Assume $w = \varepsilon$. Then, $\delta_{NSP}(z_0, w).a = a_0 = \delta_G(a_0, P_{\Sigma_G}(\varepsilon))$ and $\delta_{NSP}(z_0, w).y = y_0 = \delta_{NS}(y_0, P_{\Sigma_{NS}}(\varepsilon))$. **Induction step:** Assume that $w = v\sigma$ where the statement holds for v, i.e., $\delta_{NSP}(z_0, v).a = \delta_G(a_0, P_{\Sigma_G}(v))$ and $\delta_{NSP}(z_0, v).y = \delta_{NS}(y_0, P_{\Sigma_{NS}}(v))$. It suffices to prove that the statement holds for $v\sigma$, i.e., $\delta_{NSP}(z_0, v\sigma).a = \delta_G(a_0, P_{\Sigma_G}(v\sigma))$ and $\delta_{NSP}(z_0, v\sigma).y = \delta_{NS}(y_0, P_{\Sigma_{NS}}(v\sigma))$. Considering Definition 7, for σ enabled at $\delta_{NSP}(z_0, v)$ the following cases may occur:

$\sigma \in \Sigma_{NS} \setminus \{tick\}$, which refers to item 1) and item 5). Then, $\delta_{NSP}(z_0, v).a$ remains unchanged; $\delta_{NSP}(z_0, v\sigma).a = \delta_{NSP}(z_0, v).a = \delta_G(a_0, P_{\Sigma_G}(v)) = \delta_G(a_0, P_{\Sigma_G}(v)\varepsilon) = \delta_G(a_0, P_{\Sigma_G}(v\sigma))$, and $\delta_{NSP}(z_0, v\sigma).y = \delta_{NS}(\delta_{NS}(y_0, P_{\Sigma_{NS}}(v), \sigma) = \delta_{NS}(y_0, P_{\Sigma_{NS}}(v)\sigma) = \delta_{NS}(y_0, P_{\Sigma_{NS}}(v\sigma))$.

$\sigma \in \Sigma_G \setminus \{tick\}$, which refers to item 2) and item 3). Then, $\delta_{NSP}(z_0, v\sigma).a = \delta_G(\delta_G(a_0, P_{\Sigma_G}(v)), \sigma) = \delta_G(a_0, P_{\Sigma_G}(v)\sigma) = \delta_G(a_0, P_{\Sigma_G}(v\sigma))$, and $\delta_{NSP}(z_0, v\sigma).y$ remains unchanged; $\delta_{NSP}(z_0, v\sigma).y = \delta_{NSP}(z_0, v).y = \delta_{NS}(y_0, P_{\Sigma_{NS}}(v)\varepsilon) = \delta_{NS}(y_0, P_{\Sigma_{NS}}(v\sigma))$.

$\sigma = tick$, which refers to item 4). Then, $\delta_{NSP}(z_0, v\sigma).a = \delta_G(\delta_G(a_0, P_{\Sigma_G}(v)), \sigma) = \delta_G(a_0, P_{\Sigma_G}(v)\sigma) = \delta_G(a_0, P_{\Sigma_G}(v\sigma))$, and $\delta_{NSP}(z_0, v\sigma).y = \delta_{NS}(\delta_{NS}(y_0, P_{\Sigma_{NS}}(v)), \sigma) = \delta_{NS}(y_0, P_{\Sigma_{NS}}(v)\sigma) = \delta_{NS}(y_0, P_{\Sigma_{NS}}(v\sigma))$. **Conclusion:** By the principle of induction, the statement $(\delta_{NSP}(z_0, w).a = \delta_G(a_0, P_{\Sigma_G}(w))$ and $\delta_{NSP}(z_0, w).y = \delta_{NS}(v_0, P_{\Sigma_{NS}}(w)))$ holds for all $w \in L(NSP)$. ■

Lemma 4 (NP Transitions): Given a plant G with $x_0.a = a_0$, for any $w \in L(NP)$: $\delta_{NP}(x_0, w).a = \delta_G(a_0, P_{\Sigma_G}(w))$.

Proof: The proof is similar to the proof of Lemma 3. ■

Lemma 5 (NP Enabling Commands): Given a plant G, events from Σ_e are enabled on time in the networked plant NP (for that plant); for any $w\sigma \in L(G)$, $\sigma \in \Sigma_c$: there exists $w_0\sigma_e w_1\sigma \in L(NP)$ where $\sigma_e \in \Sigma_e$ is the enabling event of σ, $P_{\Sigma_G}(w_0w_1) = w$ and $|P_{\{tick\}}(w_1)| = N_c$.

Proof: Assume that $G' = P_{\Sigma_G \setminus \Sigma_{uc}}(G)$ is represented by $(A', \Sigma_G, \delta'_G, a'_0, A'_m)$, and let us do the proof by induction on the number of controllable events in $w \in L(G)$. **Base case:** Assume that σ is the 1^{th} controllable event enabled in G. Then, $w \in (\Sigma_{uc} \cup \{tick\})^*$. According to Assumption 1, $|P_{\{tick\}}(w)| \geq N_c$. Let say $|P_{\{tick\}}(w)| = N_c + i$ for some $i \in \mathbb{N}_0$. Then, $tick^{N_c+i}\sigma \in L(G')$. Also, assume that $w = w_i w_{N_c}$ for some w_i, w_{N_c} where $|P_{\{tick\}}(w_i)| = i$, and $|P_{\{tick\}}(w_{N_c})| = N_c$. Considering Definition 10, NP starts from $x_0 = (a_0, \delta'_G(a'_0, tick^{N_c}), [], \varepsilon)$. Then, based on item 4), $tick$ occurs in NP when it is enabled in both G and G'. For the first i *ticks*, whenever $tick$ is enabled in G, it is also enabled in G' (there are i *ticks* enabled in G' before σ occurs). Meanwhile, if there is an event ready to be observed, then based on item 5), the corresponding observed event occurs in NP which does not change the current state of G and G'. Also, based on item 3), if an uncontrollable event is enabled in G, it occurs in NP without changing the state of G'. Otherwise, $tick$ occurs in NP by being executed in both G and G'. We call this situation as G and G' are synchronized on $tick$. Therefore, it is feasible that some w_0 is executed in NP based on the execution of $tick^i$ in G' and w_i in G. Then, $\delta_{NP}(x_0, w_0).a = \delta_G(a_0, w_i)$ and $\delta_{NP}(x_0, w_0).a' = \delta'_G(a_0, tick^{N_c+i})$, and so $P_{\Sigma_G}(w_0) = w_i$. After that, since $(\delta_{NP}(x_0, w_0).a', \sigma)!$, based on item 1), σ_e occurs in NP, and (σ, N_c) is added to $\delta_{NP}(x_0, w_0).l$. Note that based on item 3) (item 5)), uncontrollable events enabled in G (events ready to be observed) can occur in between, but without loss of generality, let us assume that σ_e is enabled first, and then uncontrollable (observed) events are executed. So, w_1 will be executed in NP based on the execution of w_{N_c} in G. Therefore, $P_{\Sigma_G}(w_1) = w_{N_c}$, and $|P_{\{tick\}}(w_1)| = |P_{\{tick\}}(w_{N_c})| = N_c$. Based on item 4), by the execution of each $tick$, $\delta_{NP}(x_0, w_0\sigma_e).l$ is decreased by one. Also, σ is the only controllable event enabled in G so that $head(\delta_{NP}(x_0, w_0\sigma_e w_1).l) = (\sigma, 0)$. Then, based on item 2), σ will be executed in NP. **Induction step:** Assume that σ is the n^{th} controllable event enabled in G where the statement holds for all previous controllable events. Let us indicate the $(n-1)^{th}$ controllable event by σ^{n-1} such that $w_{n-1}\sigma^{n-1} \in L(G)$. As the statement holds for σ^{n-1}, there exists some $w_0^{n-1}\,\sigma_e^{n-1}\,w_1^{n-1}\,\sigma^{n-1} \in L(NP)$ such that $P_{\Sigma_G}(w_0^{n-1}\,w_1^{n-1}) = w_{n-1}$ and $|P_{\{tick\}}(w_1^{n-1})| = N_c$. It suffices to prove that for the next controllable event σ^n, $w_n\,\sigma^n \in L(G)$, there exists some $w_0^n\,\sigma_e\,w_1^n\,\sigma^n \in L(NP)$ with $P_{\Sigma_G}(w_0^n\,w_1^n) = w_n$ and $|P_{\{tick\}}(w_1^n)| = N_c$. Let us say $w_n = w_{n-1}\,\sigma^{n-1}w$ where $w \in (\Sigma_{uc} \cup \{tick\})^*$. Assume that $|P_{\{tick\}}(w)| = j$, and $w_{n-1} = w_i^{n-1}\,w_{N_c}^{n-1}$ where $|P_{\{tick\}}(w_i^{n-1})| = i$, and $|P_{\{tick\}}(w_{N_c}^{n-1})| = N_c$. Moreover, let us say for $w_{n-1}\,\sigma^{n-1}\,w\,\sigma \in L(G)$, there exists $tick^{N_c}\,w'_{n-1}\,\sigma^{n-1}\,tick^j\sigma^n \in L(G')$ where $|P_{\{tick\}}(w'_{n-1})| = i$. Considering Definition 10, G' synchronizes with G on executing *tick* since whenever *tick* is enabled in G', it occurs in NP only if G enables it as well. Also, uncontrollable events (observed events) occur as they are enabled in G (as the corresponding event is ready to be observed), and due to the induction assumption, all controllable events occurring in G before σ^n are enabled on time, and so they will be executed in NP. By the execution of $w_0^{n-1}\,\sigma_e^{n-1}$ in NP, $\delta_{NP}(x_0, w_0^{n-1}\,\sigma_e^{n-1}).a' = \delta'_G(a'_0, tick^{N_c}\,w'_{n-1}\,\sigma^{n-1})$, and so $\delta_{NP}(x_0, w_0^{n-1}\,\sigma_e^{n-1}).a = \delta_G(a_0, w_i^{n-1})$ (G and G' synchronize on *tick*). At this point, (before reaching σ) in G', $tick^j$ is enabled, and $w_{N_C}^{n-1}$ is enabled in G (before reaching σ^{n-1}). Then, one of the following cases may occur:

$j < N_c$. Then, assume $w_{N_c}^{n-1} = w_j^{n-1}w_{N_c-j}^{n-1}$ for some $w_j^{n-1}, w_{N_c-j}^{n-1}$ where $|P_{\{tick\}}w_j^{n-1}| = j$ and $|P_{\{tick\}}w_{N_c-j}^{n-1}| = N_c - j$. Then, the execution of $tick^j$ in G' is synchronized with the execution of w_j^{n-1} in G resulting in $w_0^{n-1}\sigma_e^{n-1}v_1 \in L(NP)$ with $|P_{\{tick\}}(v_1)| = j$ and $P_{\Sigma_G}(v_1) = w_j^{n-1}$. After that σ_e occurs in NP (as it is enabled in G') adding (σ, N_c) to l. This follows by the execution of $w_{N_c-j}^{n-1}$ in G and results in $w_0^{n-1}\sigma_e^{n-1}v_1\sigma_e^n v_2 \in L(NP)$ where $|P_{\{tick\}}(v_2)| = N_c - j$ and $P_{\Sigma_G}(v_2) = w_{N_c-j}^{n-1}$. At this point, σ^{n-1} is executed in NP following by the execution of v_3 where $P_{\Sigma_G}(v_3) = w$, and so $|P_{\{tick\}}(v_3)| = j$. This results in $w_0^{n-1}\,\sigma_e^{n-1}\,v_1\,\sigma_e^n\,v_2\,\sigma^{n-1}\,v_3 \in L(NP)$ where $P_{\{tick\}}(v_2\sigma^{n-1}v_3) = N_c - j + j = N_c$, and $head(l) = (\sigma^n, 0)$ (after the execution of σ^{n-1}, this is only (σ^n, N_c) in l, and l is decreased by one by the execution of each *tick*), and so σ^n occurs in NP. Hence, $w_0^n\,\sigma_e^n\,w_1^n\,\sigma^n \in L(NP)$ for $w_0^n = w_0^{n-1}\,\sigma_e^{n-1}\,v_1$ and $w_1^n = v_2\,\sigma^{n-1}\,v_3$ where $P_{\Sigma_G}(w_0^n\,w_1^n) = P_{\Sigma_G}(w_0^{n-1}\,\sigma_e^{n-1}\,v_1\,v_2\,\sigma^{n-1}\,v_3) = w^n$ and $|P_{\{tick\}}(w_1^n)| = N_c - j + j = N_c$.

$j > N_c$. Then, assume $w = w_{j-N_c}w_{N_c}$ for some w_{j-N_c}, w_{N_c} where $|P_{\{tick\}}w_{j-N_c}| = j - N_c$ and $|P_{\{tick\}}w_{N_c}| = N_c$. The execution of $tick^{N_c}$ in G' is synchronized with the execution of $w_{N_c}^{n-1}$ in G resulting in $w_0^{n-1}\,\sigma_e^{n-1}\,w_1^{n-1}\,\sigma_{n-1} \in L(NP)$. After that, the execution of the remaining $tick^{j-N_c}$ in G' will be synchronized with execution of w_{j-N_c} in G resulting in $w_0^{n-1}\,\sigma_e^{n-1}\,w_1^{n-1}\,\sigma^{n-1}\,v_1 \in L(NP)$ where $P_{\Sigma_G}(v_1) = w_{j-N_c}$. Then, σ_e^n occurs in NP as it is enabled in G' adding (σ^n, N_c) to l. Finally, the execution of w_{N_c} in G results in $w_0^{n-1}\sigma_e^{n-1}w_1^{n-1}\sigma^{n-1}v_1\sigma_e^n v_2 \in L(NP)$ with $P_{\Sigma_G}(v_2) = w_{N_c}$. As N_c *ticks* have passed, $head(l) = (\sigma^n, 0)$, and σ^n occurs in NP. Hence, $w_0^n\,\sigma_e^n\,w_1^n\,\sigma^n \in L(NP)$ for $w_0^n = w_0^{n-1}\,\sigma_e^{n-1}\,w_1^{n-1}\,\sigma^{n-1}\,v_1$ and $w_1^n = v_2$ where $P_{\Sigma_G}(w_0^n\,w_1^n) = w_n$ and $|P_{\{tick\}}(w_1^n)| = N_c$.

$j = N_c$. Then, after the execution of $w_0^{n-1}\sigma_e^{n-1}$ in NP, v_1 occurs in NP related to the execution of $w_{N_c}^{n-1}$ in G and w'_n in G'. At this point, σ^{n-1} is enabled in G and $head(l) = (\sigma^{n-1})$. Also, σ^n is enabled in G'. Therefore, either $\sigma_e^n\sigma^{n-1}$ or $\sigma^{n-1}\sigma_e^n$ occurs in NP both followed by the execution of some v_2 in NP such that $P_{\Sigma_G}(v_2) = w$. As N_c *ticks* have passed ($head(l) = (\sigma^n, 0)$), and σ^n is enabled in G, σ^n occurs in NP. This results in one of the following words; $w_0^{n-1}\sigma_e^{n-1}v_1\sigma^{n-1}\sigma_e^n v_2\sigma^n \in L(NP)$ or $w_0^{n-1}\sigma_e^{n-1}v_1\sigma_e^n\sigma^{n-1}v_2\sigma^n \in L(NP)$ where the statement holds in both cases as already discussed in the previous items. **Conclusion:** By the principle of induction, the statement holds for all $\sigma \in \Sigma_c$ and $w \in \Sigma_G^*$ with $w\sigma \in L(G)$. ■

Lemma 6 (NSP and NP): Consider a plant G, a networked supervisor NS (for that plant), the observation channel M, and the control channel L. The networked plant NP has the set of states X and the networked supervised plant NSP has the set of states Z. Then, for any pair of $x \in X$ and $z \in Z$ reachable through the same $w \in \Sigma_{NSP}^*$: $x.m = z.m$ and $x.l = z.l$.

Proof: Take $x \in X$, $z \in Z$, and $w \in \Sigma_{NSP}^*$ such that $x = \delta_{NP}(x_0, w)$ and $z = \delta_{NSP}(z_0, w)$. By induction on the structure of w, it is proved that $x.m = z.m$ and $x.l = z.l$. **Base case:** Assume $w = \varepsilon$. Then, $x.m = x_0.m = []$ ($x.l = x_0.l = \varepsilon$), and $z.m = z_0.m = []$ ($z.l = z_0.l = \varepsilon$). Thereto, $x.m = z.m$ and $x.l = z.l$. **Induction step:** Assume $w = v\sigma$ where the statement holds for $v \in \Sigma_{NSP}^*$ and the intermediate states reached by v so that $\delta_{NP}(x_0, v).m = \delta_{NSP}(z_0, v).m$ and $\delta_{NP}(x_0, v).l = \delta_{NSP}(z_0, v).l$. It suffices to prove that the statement holds for $v\sigma$, i.e., $\delta_{NP}(x_0, v\sigma).m = \delta_{NSP}(z_0, v\sigma).m$ and $\delta_{NP}(x_0, v\sigma).l = \delta_{NSP}(z_0, v\sigma).l$. Considering Definition 10 and Definition 7, in both operators, $\delta_{NP}(x_0, v).m$ ($\delta_{NSP}(z_0, v).m$) changes by the execution of $\sigma \in \Sigma_c \cup \Sigma_{uc} \cup tick \cup \Sigma_o$ (item 2), item 3), item 4), and item 5)), and $\delta_{NP}(x_0, v).l$ ($\delta_{NSP}(z_0, v).l$) changes by the execution of $\sigma \in \Sigma_e \cup \Sigma_c \cup tick$ (item 1), item 2), and item 4)) in a similar way. Therefore, starting from $\delta_{NP}(x_0, v)$ and $\delta_{NSP}(z_0, v)$ with $\delta_{NP}(x_0, v).m = \delta_{NSP}(z_0, v).m$ ($\delta_{NP}(x_0, v).l = \delta_{NSP}(z_0, v).l$), the execution of the same event σ results in $\delta_{NP}(x_0, v\sigma).m = \delta_{NSP}(z_0, v).m$ ($\delta_{NP}(x_0, v\sigma).l = \delta_{NSP}(z_0, v).l$). **Conclusion:** By the principle of induction, the statement ($x.m = z.m$ and $x.l = z.l$) holds for all $w \in \Sigma_{NSP}^*$, $x = \delta_{NP}(x_0, w)$ and $z = \delta_{NSP}(z_0, w)$. ■

Lemma 7 (NSP and Product): Given a plant G and a networked supervisor NS with event set Σ_{NS}. If $L(NS) \subseteq P_{\Sigma_{NS}}(L(NP))$, then $L(NSP) = L(NS||NP)$.

Proof: This is proved in two steps; 1. for any $w \in L(NSP)$: $w \in L(NS||NP)$, and 2. for any $w \in L(NS||NP)$: $w \in L(NSP)$.

1) Take $w \in L(NSP)$. By induction on the structure of w, it is proved that $w \in L(NS||NP)$. **Base case:** Assume that $w = \varepsilon$. Then, $w \in L(NS||NP)$ by definition. **Induction step:** Let $w = v\sigma$ for some $v \in \Sigma_{NSP}^*$ and $\sigma \in \Sigma_{NSP}$ where the statement holds for v, i.e., $v \in L(NS||NP)$. It suffices to prove that the statement holds for $v\sigma$, i.e., $v\sigma \in L(NS||NP)$. Due to Lemma 3, $\delta_{NSP}(z_0, v).y = \delta_{NS}(y_0, P_{\Sigma_{NS}}(v))$, $\delta_{NSP}(z_0, v\sigma).y = \delta_{NS}(\delta_{NSP}(z_0, v).y, P_{\Sigma_{NS}}(\sigma))$, $\delta_{NSP}(z_0, v).a = \delta_G(a_0, P_{\Sigma_G}(v))$, and $\delta_{NSP}(z_0, v\sigma).a = \delta_G(\delta_{NSP}(z_0, v).a, P_{\Sigma_G}(\sigma))$. Due to the definition of synchronous product (in [4]), since $\Sigma_{NS} \subseteq \Sigma_{NSP}$, one can say any $w \in L(NS||NP)$ if $w \in L(NP)$ and $P_{\Sigma_{NS}}(w) \in L(NS)$. For $v\sigma \in L(NSP)$, it is already showed that $P_{\Sigma_{NS}}(v\sigma) \in L(NS)$, and so it suffices to prove $v\sigma \in L(NP)$. For $v \in L(NS||NP)$: $v \in L(NP)$ (since $\Sigma_{NS} \subseteq \Sigma_{NSP}$). Then, due to Lemma 4, $\delta_{NSP}(z_0, v).a = \delta_G(a_0, P_{\Sigma_G}(v)) = \delta_{NP}(x_0, v).a$. Moreover, both $\delta_{NSP}(z_0, v)$ and $\delta_{NP}(x_0, v)$ are reachable through v, and so due

to Lemma 6, $\delta_{NSP}(z_0,v).m = \delta_{NP}(x_0,v).m$ and $\delta_{NSP}(z_0,v).l = \delta_{NP}(x_0,v).l$. Since $P_{\Sigma_{NS}}(v\sigma) \in L(NS)$ (for $v\sigma \in L(NSP)$, $\delta_{NS}(y_0,v\sigma)!$ due to Lemma 3), and $L(NS) \subseteq P_{\Sigma_{NS}}(L(NP))$, then one can say there exists $w' \in L(NP)$, $P_{\Sigma_{NS}}(w') = P_{\Sigma_{NS}}(v\sigma)$. Without loss of generality, assume $w' = v'P_{\Sigma_{NS}}(\sigma)$ where $P_{\Sigma_{NS}}(v') = P_{\Sigma_{NS}}(v)$. Let us complete the proof for different cases of $\sigma \in \Sigma_{NSP}$.

$\sigma \in \Sigma_e$. Then, $\delta'_G(\delta_{NP}(x_0,v).a',\sigma)!$ since $\delta_{NP}(x_0,v).a' = \delta_{NP}(x_0,v').a'$ and $\delta'_G(\delta_{NP}(x_0,v').a',\sigma)!$ ($P_{\Sigma_e\cup\{tick\}}(v) = P_{\Sigma_e\cup\{tick\}}(v')$, and due to Definition 10, $x.a'$ changes by $w \in (\Sigma_e \cup \{tick\})^*$). So, due to item 1), $\delta_{NP}(\delta_{NP}(x_0,v),\sigma)!$.

$\sigma \in \Sigma_c$. Then, $\delta_G(\delta_{NP}(x_0,v).a,\sigma)!$ since $\delta_{NP}(x_0,v).a = \delta_{NSP}(z_0,v).a$ and $\delta_G(\delta_{NSP}(z_0,v).a,\sigma)!$. Also, the condition $head(\delta_{NP}(x_0,v).l) = (\sigma,0)$ is satisfied since $\delta_{NP}(x_0,v).l = \delta_{NSP}(z_0,v).l$ and $head(\delta_{NSP}(z_0,v).l) = (\sigma,0)$ (considering Definition 7-item 2)), σ can occur only if $head(\delta_{NSP}(z_0,v).l) = (\sigma,0)$). So, due to Definition 10-item 2), $\delta_{NP}(\delta_{NSP}(z_0,v),\sigma)!$.

$\sigma \in \Sigma_{uc}$. Then, $\delta_G(\delta_{NP}(x_0,v).a,\sigma)!$ since $\delta_{NP}(x_0,v).a = \delta_{NSP}(z_0,v).a$ and $\delta_G(\delta_{NSP}(z_0,v).a,\sigma)!$. So, based on Definition 10-item 3), $\delta_{NP}(\delta_{NP}(x_0,v),\sigma)!$.

$\sigma = tick$. Then, $\delta_G(\delta_{NP}(x_0,v).a,\sigma)!$ since $\delta_{NP}(x_0,v).a = \delta_{NSP}(z_0,v).a$ and $\delta_G(\delta_{NSP}(z_0,v).a,\sigma)!$. In addition, $\delta'_G(\delta_{NP}(x_0,v).a',\sigma)!$ since $\delta_{NP}(x_0,v).a' = \delta_{NP}(x_0,v').a'$ and $\delta'_G(\delta_{NP}(x_0,v').a',\sigma)!$. Also, $(\sigma,0) \notin \delta_{NP}(x_0,v).m$ for all $\sigma \in \Sigma_a$ since $\delta_{NSP}(z_0,v).m = \delta_{NP}(x_0,v).m$ and $(\sigma,0) \notin \delta_{NSP}(z_0,v).m$ (considering Definition 7-item 4), $tick$ can occur if $(\sigma,0) \notin \delta_{NSP}(z_0,v).m$). Therefore, based on Definition 10-item 4), $\delta_{NP}(\delta_{NP}(x_0,v),\sigma)!$.

$\sigma \in \Sigma_o$. Then, $(\sigma,0) \in \delta_{NP}(x_0,v).m$ because $\delta_{NP}(x_0,v).m = \delta_{NSP}(z_0,v).m$ and $(\sigma,0) \in \delta_{NSP}(z_0,v).m$ (due to Definition 7-item 5)). So, due to Definition 10-item 5), $\delta_{NP}(\delta_{NP}(x_0,v),\sigma)!$. **Conclusion:** By the principle of induction, $w \in L(NS||NP)$ is true for any $w \in L(NSP)$.

2) Take $w \in L(NS||NP)$, by induction, it is proved that $w \in L(NSP)$ is true. **Base case:** Assume that $w = \varepsilon \in L(NS||NP)$. Then, $w \in L(NSP)$ by definition. **Induction step:** Let $w = v\sigma \in L(NS||NP)$ where the statement is true for v, i.e., $v \in L(NSP)$. It suffices to prove that the statement holds for $v\sigma$, i.e., $v\sigma \in L(NSP)$. Due to the definition of synchronous product (in [4]), since $\Sigma_{NS} \subseteq \Sigma_{NSP}$, one can say any $w \in L(NS||NP)$ if $w \in L(NP)$ and $P_{\Sigma_{NS}}(w) \in L(NS)$. Due to Lemma 4, $\delta_{NP}(x_0,v).a = \delta_G(a_0,P_{\Sigma_G}(v))$ and $\delta_{NP}(x_0,v\sigma).a = \delta_G(\delta_{NP}(x_0,v).a,\sigma)$.

Also, due to Lemma 3, for $v \in L(NSP)$, $\delta_{NSP}(z_0,v).y = \delta_{NS}(y_0,P_{\Sigma_{NS}}(v))$, and $\delta_{NSP}(z_0,v).a = \delta_G(a_0,P_{\Sigma_G}(v))$.

Moreover, since both $\delta_{NSP}(z_0,v)$ and $\delta_{NP}(x_0,v)$ are reachable through v, based on Lemma 6, $\delta_{NSP}(z_0,v).m = \delta_{NP}(x_0,v).m$ and $\delta_{NSP}(z_0,v).l = \delta_{NP}(x_0,v).l$. Now, for different cases of $\sigma \in \Sigma_{NSP}$, we prove that $\delta_{NSP}(\delta_{NSP}(z_0,v),\sigma)!$.

$\sigma \in \Sigma_e$. Then, based on the assumption, $\delta_{NS}(y_0,P_{\Sigma_{NS}}(v)\sigma)!$, and so considering Definition 7-item 1), $\delta_{NSP}(\delta_{NSP}(z_0,v),\sigma)!$.

$\sigma \in \Sigma_c$. Then, $\delta_G(\delta_{NSP}(z_0,v).a,\sigma)!$ since $\delta_{NSP}(z_0,v).a = \delta_{NP}(x_0,v).a$ and $\delta_G(\delta_{NP}(x_0,v).a,\sigma)!$. Also, the condition $head(\delta_{NSP}(z_0,v).l) = (\sigma,0)$ is satisfied since $\delta_{NSP}(z_0,v).l = \delta_{NP}(x_0,v).l$ and $head(\delta_{NP}(x_0,v).l) = (\sigma,0)$ (based on Definition 10-item 2)). Hence, considering Definition 7-item 2), $\delta_{NSP}(\delta_{NSP}(z_0,v),\sigma)!$.

$\sigma \in \Sigma_{uc}$. Then, $\delta_G(\delta_{NSP}(z_0,v).a,\sigma)!$ since $\delta_{NSP}(z_0,v).a = \delta_{NP}(x_0,v).a$ and $\delta_G(\delta_{NP}(x_0,v).a,\sigma)!$, and so considering Definition 7-item 3), $\delta_{NSP}(\delta_{NSP}(z_0,v),\sigma)!$.

$\sigma = tick$. Then, $\delta_G(\delta_{NSP}(z_0,v).a,\sigma)!$ since $\delta_{NSP}(z_0,v).a = \delta_{NP}(x_0,v).a$ and $\delta_G(\delta_{NP}(x_0,v).a,\sigma)!$. Also, $\delta_{NS}(\delta_{NS}(y_0,P_{\Sigma_{NS}}(v)),\sigma)!$ due to the assumption. Moreover, $(\sigma,0) \notin \delta_{NSP}(z_0,v).m$ for all $\sigma \in \Sigma_a$ since $(\sigma,0) \notin \delta_{NP}(x_0,v).m$ (based on Definition 10-item 4)) and $\delta_{NSP}(z_0,v).m = \delta_{NP}(x_0,v).m$. Therefore, based on Definition 7-item 4), $\delta_{NSP}(\delta_{NSP}(z_0,v),\sigma)!$.

$\sigma \in \Sigma_o$. Then, $(\sigma,0) \in \delta_{NSP}(z_0,v).m$ because $\delta_{NSP}(z_0,v).m = \delta_{NP}(x_0,v).m$ and $(\sigma,0) \in \delta_{NP}(x_0,v).m$ (due to Definition 10-item 5)). Moreover, $\delta_{NS}(\delta_{NS}(y_0,P_{\Sigma_{NS}}(v)),\sigma)!$ based on the assumption. So, due to Definition 7-item 5), $\delta_{NSP}(\delta_{NSP}(z_0,v),\sigma)!$. **Conclusion:** By the principle of induction $w \in L(NSP)$ is true for any $w \in L(NS||NP)$. ■

Corollary 1 (Lemma 7): Given a plant G and a networked supervisor NS with event set Σ_{NS} such that $L(NS) \subseteq P_{\Sigma_{NS}}(L(NP))$:

1) $L(NSP) \subseteq L(NP)$, and
2) $L_m(NSP) \subseteq L_m(NP)$.

Proof: This clearly holds since due to Lemma 7, $NSP = NS||NP$ and $\Sigma_{NS} \subseteq \Sigma_{NSP}$ ■

Lemma 8 (Finite NP): Given a plant G with a set of states A and a set of events Σ_G: NP is a finite automaton.

Proof: We need to prove that NP has a finite set of states and a finite set of events. Considering Definition 10, NP has a set of states $X = A \times Q' \times A' \times M \times L$. To prove that X is finite, it is sufficient to guarantee that A, Q', A', M and L are finite sets because as proved in [34] the Cartesian product of finite sets is finite. A is finite as the plant is assumed to be given as a finite automaton. A' is finite since for each $a' \in A'$, $a' \subseteq A$, and A is finite. $M(L)$ is finite as the maximum size of every element of M is limited to a finite number $M_{max}(L_{max})$. Moreover, $\Sigma_{NSP} = \Sigma_e \cup \Sigma_o \cup \Sigma_G$ is finite since G is a finite automaton, and so Σ_G is finite. Σ_e and Σ_o are finite since due to Definition 4, the size of Σ_e is equal to the size of Σ_c, and the size of Σ_o is equal to the size of Σ_a. ■

Lemma 9 (Nonblocking NS): The networked supervisor NS synthesized from Algorithm 1 is nonblocking.

Proof: Based on Property 3, Algorithm 1 terminates, let say after n iterations. Then, either $x_0 \in Uncon(BS(n))$ or $BS(n) = \varnothing$ where $BS(n) = BPre(NS(n) \cup BLock(NS(n)) \cup TLock(NS(n)))$. In case that $x_0 \in Uncon(BS(n))$, the algorithm gives no result. Otherwise, the algorithm gives $NS = P_{\Sigma_{NS}}(NS(n))$ where $NS(n)$ is nonblocking since $BLock(NS(n)) = \varnothing$. Moreover, due to Lemma 1, the projection preserves nonblockingness, and so NS is nonblocking. ■

Lemma 10 (TLF NS): The networked supervisor NS synthesized for a plant G using Algorithm 1 is TLF.

Proof: The proof is similar to the proof of Lemma 9. ■

Appendix B
Proofs of Properties and Theorems

A. Proof of Property 1

It suffices to prove that $w \in L(G)$ for any $w \in P_{\Sigma_G}(L(NSP))$. Take arbitrary $w \in P_{\Sigma_G}(L(NSP))$. Then, according to Definition 1, $P_{\Sigma_G}(w') = w$ for some $w' \in L(NSP)$. Then, due to Lemma 3, $\delta_{NSP}(z_0,w').a = \delta_G(a_0,P_{\Sigma_G}(w'))$ meaning that $w \in L(G)$.

B. Proof of Property 2

The proof consists of two cases:

1) for any $w \in P_{\Sigma_G}(L(NP))$: $w \in L(G)$. This is proved by induction on the structure of w. **Base case:** Assume $w = \varepsilon$. Then, $w \in L(G)$ by definition. **Induction step:** Assume that $w = v\sigma$ for some $v \in \Sigma_G^*$ and $\sigma \in \Sigma_G$ where the statement holds for v, i.e., $v \in L(G)$. It suffices to prove that the statement holds for $v\sigma$, i.e., $v\sigma \in L(G)$. Due to the projection properties, for $v\sigma \in P_{\Sigma_G}(L(NP))$, one can say there exists $v' \in \Sigma_{NSP}^*$, $P_{\Sigma_G}(v') = v\sigma$. Without loss of generality, let say $v' = v''\sigma$ where $P_{\Sigma_G}(v'') = v$. Then, due to Lemma 4, $\delta_{NP}(x_0, v'').a = \delta_G(a_0, P_{\Sigma_G}(v'')) = \delta_G(a_0, v)$, and $\delta_{NP}(\delta_{NP}(x_0, v''), \sigma).a = \delta_G(a_0, P_{\Sigma_G}(v''\sigma)) = \delta_G(\delta_G(a_0, v), \sigma)$. So, $\delta_G(\delta_G(a_0, v), \sigma)!$ and the statement holds for $v\sigma$. **Conclusion:** By the principle of induction, the statement $w \in L(G)$ holds for all $w \in P_{\Sigma_G}(L(NP))$.

2) If $max_c \leq L_{max}$, for any $w \in L(G)$: $w \in P_{\Sigma_G}(L(NP))$. This is proved by using induction on the structure of w. **Base case:** assume $w = \varepsilon$. Then $w \in P_{\Sigma_G}(L(NP))$ by definition. **Induction step:** assume that $w = v\sigma$ for some $v \in L(G)$ and $\sigma \in \Sigma_G$ where the statement holds for v, i.e., $v \in P_{\Sigma_G}(L(NP))$. It suffices to prove that $v\sigma \in P_{\Sigma_G}(L(NP))$. For $v \in P_{\Sigma_G}(L(NP))$, there exists $v' \in \Sigma_{NSP}^*$, $P_{\Sigma_G}(v') = v$ due to the projection properties. Considering Definition 10, one of the following cases may occur at $\delta_{NP}(x_0, v')$.

$\sigma \in \Sigma_{uc}$, then due to item 3), $\delta_{NP}(\delta_{NP}(x_0, v'), \sigma)!$ because $\delta_G(\delta_{NP}(x_0, v').a, \sigma)!$. Applying the projection on $v'\sigma \in L(NP)$ results in $v\sigma \in P_{\Sigma_G}(L(NP))$.

$\sigma \in \Sigma_c$, then $(\sigma, 0) \in \delta_{NP}(x_0, v').l$ since due to Lemma 5, N_c *ticks* earlier, σ_e was enabled in NP. When σ_e occurred, based on item 1), (σ, N_c) was certainly put in l as Assumption 2 holds. The occurrence of each *tick* (from N_c *ticks*) causes $l - 1$ as item 4) says. Also, the control channel is FIFO (l is a list), so even if a sequence of events have been enabled simultaneously, the ordering is preserved in l. So far, $head(\delta_{NP}(x_0, v').l) = (\sigma, 0)$ and $\delta_G(\delta_{NP}(x_0, v').a, \sigma)!$ as assumed. So, due to item 2), $v'\sigma \in L(NP)$, and $v\sigma \in P_{\Sigma_G}(L(NP))$.

$\sigma = tick$, then let us first empty $\delta_{NP}(x_0, v').m$ from any $(\sigma', 0)$ by executing $v_o \in \Sigma_o^*$. Then, $(\sigma', 0) \notin \delta_{NP}(x_0, v' v_o).m$. Also, $\delta_{NP}(x_0, v' v_o).a = \delta_{NP}(x_0, v').a$ since the execution of observed events only changes $\delta_{NP}(x_0, v').m$. $\delta_G(\delta_{NP}(x_0, v').a, tick)!$ due to the assumption, and so $\delta_G(\delta_{NP}(x_0, v' v_o).a, tick)!$. Now, as the worst case, assume that at $\delta_{NP}(x_0, v' v_o).a'$, only $v_c \in \Sigma_c^{*a}$ is enabled, and after that either *tick* occurs or nothing. Based on item 4) and item 1), this is then only v_{c_e} executed at $\delta_{NP}(x_0, v' v_o)$. $\delta_G(\delta_{NP}(x_0, v' v_o v_{c_e}).a, tick))!$, $(\sigma, 0) \notin \delta_{NP}(x_0, v' v_o v_{c_e}).m$, and $\neg\delta'_G(\delta_{NP}(x_0, v' v_o v_{c_e}).a', \sigma')!$ for all $\sigma, \sigma' \in \Sigma_a$. So, based on item 4), $v' v_o v_{c_e} tick \in L(NP)$, and so $v\,tick \in P_{\Sigma_G}(L(NP))$. **Conclusion:** By the principle of induction, the statement ($w \in P_{\Sigma_G}(L(NP))$) holds for all $w \in L(G)$.

C. Proof of Property 3

Algorithm 1 terminates if at some iteration i, $y_0 \in Uncon(bs(i))$ or $bs(i) = \varnothing$. At each iteration i, $bs(i) \subseteq Y$ since initially $bs(0) = BS(ns(0))$ where $BS(ns(0)) = BLock(ns(0)) \cup TLock(ns(0)$, and so $bs(0) \subseteq Y$ by definition. Also, $bs(i)$ is updated at line 12 to $BPre(ns(i)) \cup BS(ns(i))$ where $BPre(ns(i)) \subseteq Y$ and $BS(ns(i)) \subseteq Y$ by definition, and so $bs(i) \subseteq Y$. Since Y is a finite set, it suffices to prove that at each iteration, at least one state is removed from Y. Then, it is guaranteed that the algorithm loops finitely often. So, let's say $y_0 \notin Uncon(bs(i))$ and $bs(i) \neq \varnothing$ (because otherwise the algorithm terminates immediately). Then, there exists some state $y' \in bs(i)$. By definition this gives $y' \in Uncon(bs(i))$. Also, since at the end of each iteration, the automaton is made reachable (line 11), y' is reachable from y_0 (possibly through some intermediate states). According to line 9, at least y' is removed from Y, and so the algorithm terminates.

D. Proof of Theorem 1

We need to prove that for all $z \in Reach(z_0)$, there exists a $w \in \Sigma_{NSP}^*$ such that $\delta_{NSP}(z, w) \in Z_m$. Take $z \in Reach(z_0)$, then we need to find $w \in \Sigma_{NSP}^*$ for which $\delta_{NSP}(z, w) \in Z_m$. Let us assume that z is reachable from z_0 via $w_0 \in \Sigma_{NSP}^*$, i.e., $\delta_{NSP}(z_0, w_0) = z$. Then, due to Lemma 3, $z.y = \delta_{NS}(y_0, P_{\Sigma_{NS}}(w_0))$. Due to Lemma 9, for $z.y \in Reach(y_0)$, there exists some $v \in \Sigma_{NS}^*$ such that $\delta_{NS}(z.y, v) \in Y_m$. Moreover, due to line 11 of Algorithm 1, $L(ns(i)) \subseteq L(ns(i-1))$, and $ns(0) = NP$. Hence, $L(NS) \subseteq L(P_{\Sigma_{NS}}(NP))$ (line 15 of Algorithm 1). Then, due to the projection properties, for $P_{\Sigma_{NS}}(w_0)v \in L(NS)$, one can say there exists some $w' \in L(NP)$, $P_{\Sigma_{NS}}(w') = P_{\Sigma_{NS}}(w_0)v$ such that $\delta_{NP}(x_0, w') \in X_m$ (due to the projection properties, any state y is marked only if $y \cap X_m \neq \varnothing$). Without loss of generality, assume that $w' = w'_0 w'_1$ for some $w'_0, w'_1 \in \Sigma_{NSP}^*$ with $P_{\Sigma_{NS}}(w'_0) = P_{\Sigma_{NS}}(w_0)$ and $P_{\Sigma_{NS}}(w'_1) = v$. Let $x'_1 \in X$ be such that $\delta_{NP}(x_0, w'_0) = x'_1$, and then $\delta_{NP}(x'_1, w'_1) \in X_m$. Moreover, due to Corollary 1, $w_0 \in L(NP)$, and so $\delta_{NP}(x_0, w_0) = x_1$ for some $x_1 \in X$. So far, we have x_1, x'_1 are reachable from x_0 via w_0, w'_0, respectively, where $P_{\Sigma_{NS}}(w_0) = P_{\Sigma_{NS}}(w'_0)$. Thereto, x_1 is observationally equivalent to x'_1. Then, $x_1 \notin Uncon(BS(ns(i))$ at any iteration i because otherwise $x'_1 \in OBS(Uncon(BS(ns(i))))$, and w'_0 will be undefined ($y_0 \in Y \setminus Uncon(BS(ns(i))$, and so there exists at least a controllable event leading x_0 to x'_1 which is undefined). This is the case for all other states observationally equivalent to x_1 (because otherwise $P_{\Sigma_{NS}}(w_0) \notin L(NS)$ which contradicts the assumption). Therefore, $x_1 \notin Uncon(BS(ns(i))$ for any iteration i of the algorithm. So, at each iteration i, there exists a $w \in \Sigma_{NSP}^*$ leading x_1 to a marked state which does not become undefined because if it does, then $x_1 \in Uncon(BS(NS(i+1))$ which is a contradiction.

E. Proof of Theorem 2

We need to prove that for all $z \in Reach(z_0)$, there exists a $w \in \Sigma_{NSP}^*$ such that $\delta_{NSP}(z, w\,tick)!$. Take $z \in Reach(z_0)$, and assume z is reachable from z_0 via $w_0 \in \Sigma_{NSP}^*$, i.e., $\delta_{NSP}(z_0, w_0) = z$. Then, due to Lemma 3, $z.a = \delta_G(a_0, P_{\Sigma_G}(w_0))$ and $z.y = \delta_{NS}(y_0, P_{\Sigma_{NS}}(w_0))$. Based on Definition 7, we need to find $w \in \Sigma_{NSP}^*$ such that $\delta_G(z.a, P_{\Sigma_G}(w)\,tick)!$, $\delta_{NS}(z.y, P_{\Sigma_{NS}}(w)\,tick)!$, and $(\sigma, 0) \notin m$ for all $\sigma \in \Sigma_a$. As guaranteed by Lemma 10, NS is TLF, and so for $z.y \in Reach(y_0)$, there exists $v \in \Sigma_{NS}^*$ such that $\delta_{NS}(z.y, v\,tick)!$. Also, $L(NS) \subseteq P_{\Sigma_{NS}}(L(NP))$ (as stated before), and so from the projection properties, one can say there exists $v' \in \Sigma_{NSP}^*$, $P_{\Sigma_{NS}}(v') = v$, $\delta_{NP}(x, v'\,tick)!$. Let us take $w = v'$ for which we already know $\delta_{NS}(y, P_{\Sigma_{NS}}(w)\,tick)!$. Also, $(\sigma, 0) \notin m$ for all $\sigma \in \Sigma_a$ because otherwise Definition 10-item 4) could not be satisfied. It now suffices to prove $\delta_G(z.a, P_{\Sigma_G}(w)\,tick)!$. As Property 2 says, $P_{\Sigma_G}(L(NP)) \subseteq L(G)$, and so $P_{\Sigma_G}(w)\,tick \in L(G)$ for $w\,tick \in L(NP)$.

F. Proof of Theorem 3

We need to prove that if we take any $w \in L(NSP)$ and $u \in \Sigma_{uc} \cup \{tick\}$ such that $P_{\Sigma_G}(w)u \in L(G)$. Then, $wu \in L(NSP)$ for $u \in \Sigma_{uc}$, and for $u = tick$ when there does not exist any $\sigma_f \in \hat{\Sigma}_{for} \cup \Sigma_o$ such that $w\sigma_f \in L(NSP)$.

Take $w \in L(NSP)$ and $u \in \Sigma_{uc}$. From Lemma 3, $\delta_{NSP}(z_0, w).a = \delta_G(a_0, P_{\Sigma_G}(w))$. Based on Definition 7-item 2), u occurs only if it is enabled by G. So, $\delta_{NSP}(\delta_{NSP}(z_0, w), u)!$ since $\delta_G(\delta_{NSP}(z_0, w).a, u)!$ due to the assumption.

Take $u = tick$ where $\nexists_{\sigma \in \hat{\Sigma}_{for} \cup \Sigma_o}\ w\sigma \in L(NSP)$. Considering Definition 7-item 4), *tick* occurs in *NSP* after w if the following conditions hold; 1. $P_{\Sigma_G}(w)\,tick \in L(G)$, 2. $P_{\Sigma_{NS}}(w)\,tick \in L(NS)$, and 3. $\nexists \sigma \in \Sigma_o, \delta_{NSP}(z_0, w\sigma)!$. The first and the last conditions hold based on the assumption. So, we only need to prove $P_{\Sigma_{NS}}(w)\,tick \in L(NS)$. Due to Corollary 1, for $w \in L(NSP)$: $w \in L(NP)$. Due to Property 2, for $P_{\Sigma_G}(w)\,tick \in L(G)$, there exists $w' \in L(NP)$, $P_{\Sigma_G}(w') = P_{\Sigma_G}(w)$ such that $w'.tick \in L(NP)$. Considering Definition 10, $w\,tick \in L(NP)$ for the following reasons; 1. $\delta_{NP}(x_0, w).a = \delta_{NP}(x_0, w').a$ and $\delta_{NP}(x_0, w).a' = \delta_{NP}(x_0, w').a'$ since $P_{\Sigma_G}(w') = P_{\Sigma_G}(w)$. Hence, $\delta_G(\delta_{NP}(x_0, w).a, tick)!$ and $\delta_G(\delta_{NP}(x_0, w).a', tick)!$ (since $\delta_{NP}(x_0, w'\,tick)!$). 2. $m \in M$ changes only by the execution of $\sigma \in \Sigma_G$. So, $\delta_{NP}(x_0, w).m = \delta_{NP}(x_0, w').m$ since $P_{\Sigma_G}(w') = P_{\Sigma_G}(w)$. Also, $(\sigma, 0) \notin \delta_{NP}(x_0, w').m$ for any $\sigma \in \Sigma_a$ since $\delta_{NP}(x_0, w'\,tick)!$, and so $(\sigma, 0) \notin \delta_{NP}(x_0, w).m$ for any $\sigma \in \Sigma_a$. Due to the assumption, $w\sigma \notin L(NSP)$ for $\sigma \in \Sigma_{for} \cup \Sigma_e \cup \Sigma_o$. Also, due to Theorem 7, $NSP = NS||NP$. In case that $w\sigma \in L(NP)$ for some $\sigma \in \Sigma_{for} \cup \Sigma_o$, then, due to line 5 of Algorithm 1, it could not be disabled by *NS*. Also, if $w\sigma \in L(NP)$ for some $\sigma \in \Sigma_e$ where both *tick* and σ become disabled by *NS*, then by definition, $\delta_{NP}(x_0, w) \in BPre(NS)$ and will be removed which violates the assumption ($w \in L(NSP)$). Hence, $w\,tick \in L(NP)$ and *tick* does not become disabled by Algorithm 1, and so $P_{\Sigma_{NS}}(w)\,tick \in L(NS)$.

G. Proof of Theorem 4

To prove that *NS* is (timed networked) maximally permissive for G, we need to ensure that for any other proper networked supervisor (say NS') in the same NSC framework (with event set Σ_{NS}): $P_{\Sigma_G}(L(NS'_{N_c}\|_{N_o} G)) \subseteq P_{\Sigma_G}(L(NSP))$. First, assume that $L(NS') \nsubseteq P_{\Sigma_{NS}}(L(NP))$. Then, any extra transition of NS' that is not included in $P_{\Sigma_{NS}}(L(NP))$ does not add any new transition to $P_{\Sigma_G}(L(NS'_{N_c}\|_{N_o} G))$. Let say $v\sigma \in L(NS')$ and $v \in P_{\Sigma_{NS}}(L(NP))$, but $v\sigma \notin P_{\Sigma_{NS}}(L(NP))$ for $\sigma \in \Sigma_{NS}$. Also, there exists $w \in L(NS'_{N_c}\|_{N_o} G)$ with $P_{\Sigma_{NS}}(w) = v$. If $\sigma = tick$, then σ cannot be executed in $NS'_{N_c}\|_{N_o} G$ because based on Definition 7-item 4), *tick* should be enabled by G which is not the case; *tick* is not enabled in *NP*, and so due to Property 2, it is not enabled in G. If $\sigma \in \Sigma_o$, then it does not matter if σ occurs in $NS'_{N_c}\|_{N_o} G$ because it does not change $P_{\Sigma_G}(L(NS'_{N_c}\|_{N_o} G))$. If $\sigma \in \Sigma_e$, then as Lemma 5 says, *NP* enables all enabling events of Σ_c that are executed in the plant on time (N_c *ticks* ahead). So, any extra enabling event by NS' will not be executed by the plant, and so it does not enlarge $P_{\Sigma}(L(NS'_{N_c}\|_{N_o} G))$. Therefore, we continue the proof for the case that $L(NS') \subseteq P_{\Sigma_{NS}}(L(NP))$ (where Lemma 7 and Corollary 1 hold for NS'). Take an arbitrary $w \in P_{\Sigma_G}(L(NS'_{N_c}\|_{N_o} G))$, it suffices to prove that $w \in P_{\Sigma_G}(L(NSP))$. Let say $NS'_{N_c}\|_{N_o} G = (z'_0, \Sigma_{NSP}, \delta_{NS'P}, Z'_m)$. For $w \in P_{\Sigma_G}(L(NS'_{N_c}\|_{N_o} G))$, due to the projection properties, there exists $v' \in L(NS'_{N_c}\|_{N_o} G)$ such that $P_{\Sigma_G}(v') = w$ where $\delta_{NS'P}(z'_0, v')$ is a TLF and non-blocking state (NS' is proper due to the assumption). Also, any uncontrollable active event/non-preemptable *tick* enabled at $\delta_G(a_0, w)$ is enabled at $\delta_{NS'P}(z_0, v')$, and it leads to a nonblobking and TLF state. Based on Lemma 3, $P_{\Sigma_{NS}}(v') \in L(NS')$ for $v' \in L(NS'_{N_c}\|_{N_o} G)$, and due to Corollary 1, $v' \in L(NP)$. Moreover, due to Lemma 7, $L(NS'_{N_c}\|_{N_o} G) = L(NS'||NP)$, so regarding the definition of synchronous product, for any $w' \in L(NP)$ and $P_{\Sigma_{NS}}(w') = P_{\Sigma_{NS}}(v')$: $w' \in L(NS'_{N_c}\|_{N_o} G)$. $\delta_{NS'P}(z'_0, w')$ is a TLF and nonblocking state because $NS'_{N_c}\|_{N_o} G$ is nonblocking and TLF due to the assumption. Also, any uncontrollable active event or non-preemptable *tick* enabled at w' leads to a nonblocking and TLF state since NS' is controllable for G by the assumption. Therefore, one can say $\delta_{NP}(x_0, v') \notin OBS(Uncon(BS(NP))$. Considering Algorithm 1, initially, $ns(0) = NP$ where $v' \in L(NP)$ and $\delta_{NS}(y_0, v') \notin OBS(Uncon(BS(ns(0)))$. The last statement holds for any iteration of the algorithm until the last one (say n) so that $\delta_{NS}(y_0, v') \notin OBS(Uncon(BS(NS(n)))$ because otherwise all $y \in OBS(Uncon(BS(NS(n)))$ are removed (based on line 6), and so $P_{\Sigma_{NS}}(v') \notin L(NS)$ because it leads $NS||NP$ ($NS||NP = NS(n)$) to a state in $Uncon(BS(NS(n)))$. Then, based on Lemma 7, *NSP* becomes blocking/time-lock/uncontrollable which violates the assumption. Hence, (considering line 6) v' is not undefined by Algorithm 1, and so $P_{\Sigma_{NS}}(v') \in L(NS)$. Based on Lemma 7, $L(NSP) = L(NS||NP)$. $P_{\Sigma_{NS}}(v') \in L(NS)$ and $v' \in L(NP)$, so $v' \in L(NSP)$ where applying the projection on Σ_G gives $w \in P_{\Sigma_G}(L(NSP))$.

H. Proof of Theorem 5

To simplify, let us denote $G||R^{\perp}$ by G^t, the networked plant $\Pi(G^t, N_c, N_o, L_{max}, M_{max})$ by NP^t and the networked supervised plant $NS_{N_c}\|_{N_o} G^t$ by NSP^t. We need to prove that if we take any $w \in P_{\Sigma_{NSP} \cap \Sigma_R}(L(NSP^t))$, then $w \in P_{\Sigma_{NSP} \cap \Sigma_R}(L(R))$. In our setting, $\Sigma_{NSP} \cap \Sigma_R = \Sigma_R$ since $\Sigma_{NSP} = \Sigma_e \cup \Sigma \cup \Sigma_o$ and $\Sigma_R \subseteq \Sigma_G$. Hence, it suffices to prove that for any $w \in P_{\Sigma_R}(L(NSP^t))$: $w \in L(R)$. Take $w \in P_{\Sigma_R}(L(NSP^t))$, then due to Definition 1, there exists $w' \in L(NSP^t)$ such that $P_{\Sigma_R}(w') = w$. Also, based on Property 1, $P_{\Sigma_G}(L(NSP^t)) \subseteq L(G^t)$, and so $P_{\Sigma_G}(w') \in L(G||R^{\perp})$. Applying the projection on Σ_R gives $P_{\Sigma_R}(w') \in L(R^{\perp})$. For $w \in P_{\Sigma_R}(L(NSP^t)) \cap L(R^{\perp})$, $w \in L(R)$ since the blocking state q_d added to $G||R$ to make $G||R^{\perp}$ is removed by *NS* as guaranteed by Theorem 1.

References

[1] R. A. Gupta and M.-Y. Chow, “Networked control system: Overview and research trends,” *IEEE Transactions on Industrial Electronics*, vol. 57, no. 7, pp. 2527–2535, 2010.

[2] W. M. H. Heemels, A. R. Teel, N. Van de Wouw, and D. Nesic, "Networked control systems with communication constraints: Tradeoffs between transmission intervals, delays and performance," *IEEE Transactions on Automatic Control*, vol. 55, no. 8, pp. 1781–1796, 2010.

[3] P. Antsaklis and J. Baillieul, "Special issue on technology of networked control systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 5–8, 2007.

[4] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[5] W. M. Wonham, K. Cai *et al.*, *Supervisory control of discrete-event systems*. Springer.

[6] B. A. Brandin and W. M. Wonham, "Supervisory control of timed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 39, no. 2, pp. 329–342, 1994.

[7] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM Journal on Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[8] P. Ramadge and W. Wonham, "Supervisory control of a class of discrete event processes," in *Analysis and Optimization of Systems*. Springer, 1984, pp. 475–498.

[9] P. Xu, S. Shu, and F. Lin, "Nonblocking networked control of discrete event systems," in *2017 Chinese Automation Congress (CAC)*, 2017, pp. 1911–1916.

[10] X. Yin and S. Lafortune, "Synthesis of maximally permissive supervisors for partially-observed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 61, no. 5, pp. 1239–1254, 2015.

[11] S. Balemi, "Communication delays in connections of input/output discrete event processes," in *1992 31st IEEE Conference on Decision and Control*. IEEE, 1992, pp. 3374–3379.

[12] S.-J. Park and K.-H. Cho, "Delay-robust supervisory control of discrete-event systems with bounded communication delays," *IEEE Transactions on Automatic Control*, vol. 51, no. 5, pp. 911–915, 2006.

[13] S.-J. Park, "Robust and nonblocking supervisory control of nondeterministic discrete event systems with communication delay and partial observation," *International journal of control*, vol. 85, no. 1, pp. 58–68, 2012.

[14] F. Lin, "Control of networked discrete event systems: Dealing with communication delays and losses," *SIAM Journal on Control and Optimization*, vol. 52, no. 2, pp. 1276–1298, 2014.

[15] S. Shu and F. Lin, "Deterministic networked control of discrete event systems with nondeterministic communication delays," *IEEE Transactions on Automatic Control*, vol. 62, no. 1, pp. 190–205, 2017.

[16] Z. Liu, X. Yin, S. Shu, and S. Li, "Online supervisory control of networked discrete-event systems with control delays," in *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, 2019, pp. 6706–6711.

[17] A. Rashidinejad, M. Reniers, and M. Fabian, "Supervisory control of discrete-event systems in an asynchronous setting," in *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2019, pp. 494–501.

[18] A. Rashidinejad, M. Reniers, and L. Feng, "Supervisory control of timed discrete-event systems subject to communication delays and non-FIFO observations," *IFAC-PapersOnLine*, vol. 51, no. 7, pp. 456 – 463, 2018, 14th IFAC Workshop on Discrete Event Systems WODES 2018.

[19] B. Zhao, F. Lin, C. Wang, X. Zhang, M. P. Polis, and L. Y. Wang, "Supervisory control of networked timed discrete event systems and its applications to power distribution networks," *IEEE Transactions on Control of Network Systems*, vol. 4, no. 2, pp. 146–158, 2017.

[20] S. Shu and F. Lin, "Supervisor synthesis for networked discrete event systems with communication delays," *IEEE Transactions on Automatic Control*, vol. 60, no. 8, pp. 2183–2188, 2015.

[21] M. V. S. Alves, L. K. Carvalho, and J. C. Basilio, "Supervisory control of timed networked discrete event systems," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, 2017, pp. 4859–4865.

[22] S. Shu and F. Lin, "Predictive networked control of discrete event systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 9, pp. 4698–4705, 2017.

[23] F. Lin, *Modeling and Control of Networked Discrete-Event Systems*. Wiley Encyclopedia of Electrical and Electronics Engineering, 2020, pp. 1–27.

[24] Y. Zhu, L. Lin, S. Ware, and R. Su, "Supervisor synthesis for networked discrete event systems with communication delays and lossy channels," in *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, 2019, pp. 6730–6735.

[25] S.-J. Park and K.-H. Cho, "Nonblocking supervisory control of timed discrete event systems under communication delays: The existence conditions," *Automatica*, vol. 44, no. 4, pp. 1011 – 1019, 2008.

[26] C. Miao, S. Shu, and F. Lin, "Predictive supervisory control for timed discrete event systems under communication delays," in *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, 2019, pp. 6724–6729.

[27] S. Ware and R. Malik, "The use of language projection for compositional verification of discrete event systems," in *2008 9th International Workshop on Discrete Event Systems*. IEEE, 2008, pp. 322–327.

[28] J. E. Hopcroft, R. Motwani, and J. D. Ullman, "Introduction to automata theory, languages, and computation," *ACM SIGACT News*, vol. 32, no. 1, pp. 60–65, 2001.

[29] J. Carroll and D. Long, *Theory of Finite Automata with an Introduction to Formal Languages*. Prentice-Hall, Inc., 1989.

[30] S. Takai and T. Ushio, "A new class of supervisors for timed discrete event systems under partial observation," *Discrete Event Dynamic Systems*, vol. 16, no. 2, pp. 257–278, 2006.

[31] K. Cai, R. Zhang, and W. M. Wonham, "Relative observability and coobservability of timed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 61, no. 11, pp. 3382–3395, 2016.

[32] H. Flordal, R. Malik, M. Fabian, and K. Åkesson, "Compositional synthesis of maximally permissive supervisors using supervision equivalence," *Discrete Event Dynamic Systems*, vol. 17, no. 4, pp. 475–504, 2007.

[33] A. Rashidinejad, P. van der Graaf, M. Reniers, and M. Fabian, "Non-blocking supervisory control of timed automata using forcible events," in *15th International Workshop on Discrete Event Systems (WODES 2020)*. IEEE, 2020, accepted. [Online]. Available: https://michelreniers.files.wordpress.com/2020/06/wodes20_0055_fi.pdf

[34] T. Jech, *Set theory*. Springer Science & Business Media, 2013.

Aida Rashidinejad received the M.Sc. degree in electrical-control engineering from Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran, in 2014. She is currently working towards PhD degree in mechanical engineering-control systems from Eindhoven University of Technology, Eindhoven, The Netherlands. Her current research interests include supervisory control synthesis, networked control, and cyber-physical systems.

Michel Reniers (S'17) is currently an Associate Professor in model-based engineering of supervisory control at the Department of Mechanical Engineering at TU/e. He has authored over 100 journal and conference papers. His research portfolio ranges from model-based systems engineering and model-based validation and testing to novel approaches for supervisory control synthesis. Applications of this work are mostly in the areas of cyber-physical systems.

Martin Fabian is Professor in Automation and Head of the Automation Research group at the Department of Electrical Engineering, Chalmers University of Technology. His research interests include formal methods for automation systems in a broad sense, merging the fields of Control Engineering and Computer Science. He has authored more than 200 publications, and is co-developer of the formal methods tool Supremica, which implements several state-of-the-art algorithms for supervisory control synthesis.