Security Analysis for Distributed IoT-Based Industrial Automation

Vuk Lesi, Student Member, IEEE, Zivana Jakovljevic, Member, IEEE, and Miroslav Pajic, Senior Member, IEEE

Abstract—With ever-expanding computation and communication capabilities of modern embedded platforms, Internet of Things (IoT) technologies enable development of Reconfigurable Manufacturing Systems—a new generation of highly modularized industrial equipment suitable for highly-customized manufacturing. Sequential control in these systems is largely based on discrete events, while their formal execution semantics is specified as Control Interpreted Petri Nets (CIPN). Despite industry-wide use of programming languages based on the CIPN formalism, formal verification of such control applications in the presence of adversarial activity is not supported. Consequently, in this paper we focus on security-aware modeling and verification challenges for CIPN-based sequential control applications. Specifically, we show how CIPN models of networked industrial IoT controllers can be transformed into Time Petri Net (TPN)-based models, and composed with plant and security-aware channel models in order to enable system-level verification of safety properties in the presence of network-based attacks. Additionally, we introduce realistic channel-specific attack models that capture adversarial behavior using nondeterminism. Moreover, we show how verification results can be utilized to introduce security patches and motivate design of attack detectors that improve overall system resiliency, and allow satisfaction of critical safety properties. Finally, we evaluate our framework on an industrial case study.

Note to Practitioners—Our main goal is to provide formal security guarantees for distributed sequential controllers. Specifically, we target smart automation controllers geared towards Industrial IoT applications, that are typically programmed in C/C++, and are running applications originally designed in e.g., GRAFCET (IEC 60848)/SFC (IEC 61131-3) automation programming languages. Since existing tools for design of distributed automation do not support system-level verification of relevant safety properties, we show how security-aware transceiver and communication models can be developed and composed with distributed controller models. Then, we show how existing tools for verification of Time Petri Nets can be used to verify relevant properties including safety and liveness of the distributed automation system in the presence of network-based attacks. To provide an end-to-end analysis as well as security patching, results of our analysis can be used to deploy suitable firmware updates during the stage when executable code for target controllers (e.g., in C/C++) is generated based on GRAFCET/SFC control models. We also show that security guarantees can be improved as the relevant safety/liveness properties can be verified after corresponding security patches are deployed. Finally, we show applicability of our methodology on a realistic distributed pneumatic manipulator.

Primary and Secondary Keywords—Primary Topics: Sequential control systems, Secure distributed automation, Industrial Internet of Things; Secondary Topics: Petri nets, Non-deterministic analysis

I. INTRODUCTION

Advanced capabilities of smart Internet of Things (IoT) devices have lead to their widespread adoption in industrial automation system, rapidly advancing reconfigurable manufacturing [1]; the rise of the fourth industrial revolution, known as Industry 4.0 [2], introduces the new era of highly-customized (rather than highly-serialized) manufacturing [3]. In this vision, manufacturing resources are highly modularized, providing the necessary flexibility to adapt to dynamical market demands. Efficient structural and functional changes are supported by Reconfigurable Manufacturing Systems (RMS) that can be configured ad-hoc with little or zero downtime [4]. The foundation of RMS are modules controlled by smart Industrial IoT (IIoT)-enabled controllers. IIoT endpoints (sometimes referred to as industrial assets) are heterogeneous by definition—they represent multi-vendor components whose deployment environment dynamically changes depending on the process needs and current configuration of RMS. Furthermore, a plethora of different communication technologies (wired and wireless) and protocols are employed [5]. Seamless reconfiguration, integration and reliable functioning of RMS requires that components are highly autonomous. Specifically, they must be capable of seamlessly communicating with each other using compatible protocols (integrability), exchanging both low-level control-related and high-level process-bound information (interoperability) and to interact with each other in different ways to enable operation in a plethora of configurations (composability) [6].

Reconfigurability is naturally supported by distributed control architectures; conventionally centralized controllers are responsible of all aspects of control—from low-level event signaling, to high-level coordination. Their complexity hinders reconfigurability both from the hardware perspective (i.e., requiring component re-wiring), and the software aspect (i.e., having to ensure the control software is aware of and functions correctly under the new hardware configuration). Thus, the new generation of smart manufacturing resources must exploit not only functionally-required components (such as sensors and actuators) but also intrinsic computation and communication capabilities of IIoT-enabled controllers in order to enable a higher level of automation and autonomy. Control distribution enables decoupling of fine-grained details about how control over the specific physical resource is performed, from the resources coordination problem which only needs to worry about what the manufacturing resources are capable of performing.

However, the networked nature of the new generation of distributed automation systems makes them susceptible to network-based attacks [7], similar to security vulnerabilities reported in other cyber-physical systems domains (e.g., [8]).
security-aware modeling. Thus, for the first phase of our security analysis, we introduce methods for automatic transformation of domain-specific CIPN-based controller specifications (i.e., designs) into TPN-compliant representations. These TPN models enable closed-loop system modeling and analysis, by composing them with corresponding non-deterministic plant and security-aware communication channel models; we show how such security-aware models can be developed with the desired level of abstraction that allows us to capture impacts of attacks on automation performance. While our framework generally supports any communication channel implementation, we focus on the IEEE 802.15.4-based implementation featured in our evaluation setup.

In Phase 2, we employ open verification tools (e.g., [16]) to perform system-wide verification of safety and QoC-relevant properties in the presence of attacks, based on the aforementioned security-aware closed-loop system model; it is important to highlight that we make no assumptions about the attacker’s choice among all possible malicious actions nor the times when they (i.e., attack actions) may occur. By enabling security analysis within the same family of formalisms (i.e., using a formalism that is closely related to the formalism used to design controllers), we provide convenient domain-specific interpretation of analysis results. This allows us to exploit verification results in Phase 3 to orchestrate security patches in code generation which is performed based on original CIPN-based models. Finally, we show the applicability of our methodology on a real-world industrial case study—-a security analysis of an IIoT-enabled manipulator system.

Specifically, the contributions of this work are as follows:

- Security-aware framework for verification of system-level properties for distributed discrete-event controllers (based on CIPNs) in the presence of network-based attacks;
- TPN-based non-deterministic modeling of network-based attacks on distributed controller communication, with emphasis on capturing impacts on automation performance;
- Extension of the control software development cycle from security-aware analysis to firmware patching, in order to ensure correct operation in the presence of attacks;
- Full-stack proof-of-concept case study based on industry-grade components demonstrating applicability of the developed secure automation framework.

This paper is organized as follows. Sec. II gives an overview of relevant related work, while Sec. III provides the problem definition with emphasis on distributed IIoT-based automation. Sec. IV introduces TPN-based security-aware modelings and Sec. V derives the security-aware communication model. Specification and verification of relevant formal properties is presented in Sec. VI, as well as the loop closure from verification to code generation to include security patches and improve system resiliency. Industrial case studies are discussed in Sec. VII, before concluding remarks (Sec. VIII).

II. RELATED WORK

In [17], a model-based approach for simulating attacks on CPS is presented, but no formal verification is supported and experimental results are obtained based on specific attack implementations. In [18], additional formal security assessment of industrial CPS controllers is performed, but analysis
remains constrained to high-level vulnerabilities at the level of functional models. On the other hand, a comprehensive formal security analysis of wireless IoT communications under a specific attack model is presented in [19], but no relations to implications for Quality-of-Control of the underlying physical process are considered.

Security analysis techniques for other IoT domains have also been recently proposed. In [20], smart home IoT applications are formally surveyed for anomalous behaviors. However, formal adversarial models and implications of security vulnerabilities on system operation, such as QoC and safety in the presence of attacks, are not considered. Similarly, [21] introduces a dynamic policy-based enforcement system for securing against unauthorized and unwanted control scenarios, but focuses only on architectures and platforms for consumer IoT applications mostly in smart home automation. In [12], an SMT-based framework for IoT security analysis is presented; yet, only abstract threat models are used, and the software architecture of IoT nodes is masked by behavioral modeling.

Note that Petri Nets (PNs) have been used for security-aware modeling and analysis. For example, penetration analysis using attack trees was formalized through PNs (e.g., [22], [23]). Coordinated cyber-physical attack modeling for smart grids was done in [24], but high-level attack scenarios were modeled, and the structure of the system components was coarsely abstracted. Modeling of risks and vulnerabilities towards avoidance and discovery for Unix-like software was performed (e.g., [25]) but without specifics of the underlying software architecture. [26], [27] adopt stochastic PN-based attack models for CPS threats, while in [28], authors formulate a framework for formal reliability analysis of networked IoT sequential control applications based on CIPNs. On the other hand, [29] deals with fault detection in systems modeled by PNs. However, fault/failure models are limited to stochastic behaviors that cannot accurately capture adversarial actions (as described in [7]). While cooperation and communication protocols were modeled with PNs (e.g., [30], [31]), to the best of our knowledge, nondeterminism in Petri nets was not exploited for adversarial modeling.

III. MOTIVATING EXAMPLE AND PROBLEM DESCRIPTION

We first introduce CIPNs and the mother formalism of PNs, before presenting an illustrative distributed control application, used as a running example in this work to highlight security vulnerabilities caused by automation (i.e., control) distribution.

Petri Nets (PNs): A Petri net is a 5-tuple $PN = (P, T, F, W, M_0)$, where $P = \{P_1, \ldots, P_m\}$ is a set of places (represented by circles), $T = \{T_1, \ldots, T_n\}$ is a set of transitions (represented by bars) such that $P \cup T \neq \emptyset$ and $P \cap T = \emptyset$, while $F \subseteq \{P \times T\} \cup \{T \times P\}$ is the set of arcs between places and transitions (no arc connects two places or two transitions). $W$ is the vector of arc cardinalities which determines how many tokens are removed/deposited over specific arcs upon firing of corresponding transitions. The PN’s state is defined by its marking, i.e., distribution of tokens (captured by dots inside places); $M_0$ is the initial marking (i.e., the initial token distribution). Functionally, current PN marking determines the system’s state, while transition firing (i.e., token flow) represents a state change. A formal description of the PN semantics can be found in [32].

Distributed Automation with CIPNs: CIPNs are a version of PNs where arc cardinality is fixed to 1 and the initial marking $M_0$ may initialize only one place with a token. In CIPNs, transitions’ firing can be conditioned by system inputs (i.e., sensors) in the form $sensor=value$, while actuation commands can be associated with places in the form $actuator=command$. For distributed automation, functionality of each local controller (LC) is captured by the corresponding CIPN. For (event) information exchange between LCs, places of a CIPN may invoke the communication API exposed by the LC runtime environment; this is denoted as $Send(signal,value)$ for broadcast or $Send({dest1,dest2,...},signal,value)$ for uni/multicast transmissions. Dually, the receiving LC can condition its transitions with statements similar to conditioning on locally connected sensors (i.e., as $signal=value$) [15].

Formally, a CIPN is a 6-tuple $CIPN = (P, T, F, C, A, M_0)$ where $P$, $T$, $F$, and $M_0$ are defined as for PNs; $C = \{C_1, \ldots, C_n\}$ is a set of logical conditions enabling synchronization of the controller with sensors by guarding corresponding transitions in the CIPN model; $A = \{A_1, \ldots, A_m\}$ is a set of actions on actuator outputs that are allocated to places; formal CIPN semantics is available in [33]. By its definition, CIPN semantics is deterministic (does not support nondeterminism due to CIPNs use to only model controllers), which needs to be ensured during model design [34].

Distributed CIPN-based controller models are obtained directly by design, or by distributing existing (i.e., centralized) controllers (e.g., as done in [15]). Fig. 2 shows a simple control application, built with two wireless nodes, that we will use as
a running example in this work.

**Example 1:** Consider a simple application from Fig. 2. Control over the physical system (Fig. 2(a)) is performed by the conveyor monitor (LC1) and the pick & place station controller (LC2). Two sensors (e.g., proximity, retro-reflective) locally connected to LC1 overlook two parallel incoming conveyor belts; they sense if a workpiece is ready to be picked from either of the conveyors and placed on the third, outgoing conveyor. The CIPN-based controllers for LC1 and LC2 are shown in Fig. 2(b-c). Initially, LC1 is in state Pcm_Init where it is waiting for either of its sensors to indicate workpiece presence (i.e., transition Tcm_Pres1 / Tcm_Pres2 is conditioned by the sensing event Pres1=1 / Pres2=1). Upon detection of a workpiece (i.e., when one of Pres1=1 / Pres2=1 is satisfied, and thus transition Tcm_Pres1 / Tcm_Pres2 is enabled), LC1 sends a message to LC2 (via API call Send(Pick,1) / Send(Pick,2) in place Pcm_TxCtrl_Pick1/...Pick2); the message indicates which conveyor has a workpiece ready to be picked. LC1 then waits for completion of the pick & place operation.

Concurrently, LC2's initial state is Pctr1_Init where the pick & place station is commanded to halt (by PP_Act=0). Once the signal Pick is received from LC1, based on its value the token in LC2 model transitions over Tctrl_wfPick1 / Tctrl_wfPick2 into place Pctrl_PsP1 / Pctrl_PsP2 where the corresponding actuation command is given to the pick & place station (i.e., PP_Act=1 / PP_Act=2). After completion of the pick & place operation, condition P&P_Complete==1 is satisfied, allowing the LC2 to transition over Tctrl_P&Pcomplete / Tctrl_P&Pcomplete to Pctr1_Ret where it commands the pick & place station to return to home position (by PP_Act=-1). LC2 waits for completion of the pick & place station return stroke (when Ret_Complete==1 evaluates to true); after it transitions back into Pctr1_SendCMfin, where it signals LC1 that the workcycle is complete. LC2 transitions over Tctrl_RetInit back into the initial place Pctr1_Init. Finally, conveyor monitor LC1 can also return to its initial state (formally, the token is deposited back into Pbm_Init over Tbm_Init over Tbm_Init1 / Tbm_Init2), as the condition Complete==1 is satisfied.

The CIPN-based control models from Example 1 assume ideal communication (i.e., packet delivery), without unpredictable channel behaviors. For instance, consider an adversary with network access that mounts an impersonation (i.e., spoofing or masquerade [10]) attack when the right conveyor belt contains a workpiece.

IV. TPN-BASED AUTOMATION MODELING

TPNs extend PNs by introducing timed transitions; in a TPN, every transition is characterised by an interval $[l_f, \bar{l_f}])$ or $[l_f, \bar{l_f})$, where $l_f$ and $\bar{l_f}$ are the lower/upper bound on the transition firing times, which may be zero or infinity—time interval next to immediate transitions (i.e., with zero firing time) is not specified. Also, the
firing times are defined relative to the moment of transition enabling, without any assumptions on their distribution. This enables modeling of timed properties of real-time control software [40], [41], [42]. In addition, by supporting non-determinism, TPNs facilitate attack modeling that cannot be accurately done with deterministic or stochastic models.

Therefore, we transform formal distributed control specifications expressed by CIPN, \( i = 1, \ldots, N \), into TPN-compatible models TPN\(^{ctrl}_i\), \( i = 1, \ldots, N \). We then compose these models with plant models TPN\(^{plant}_i\), \( i = 1, \ldots, N \), and security-aware communication channel models TPN\(^{channel}_i\), \( j = 1, \ldots, M^2 \) which enables us to reason about system-level safety and security properties under the modeled adversarial influences. Since both CIPNs and TPNs originate from PNs, the translation from CIPN, \( i = 1, \ldots, N \) controller models to TPN\(^{ctrl}_i\), \( i = 1, \ldots, N \) is direct for all places and transitions except where platform-implemented API is called, i.e.,

1. Places handling actuation, and transitions handling sensing by issuing I/O API calls (actuator=value and sensor==value, respectively),
2. Places handling transmissions over a shared channel, and transitions handling receiving of communication signals using API calls (Send(destination,signal) and signal==value, respectively),
3. Places calling other platform-dependent API, such as request for execution delays.

These CIPN constructs, which directly rely on the underlying platform used to implement the controller, must be explicitly modeled as nets that capture: 1) interaction between LCs TPN\(^{ctrl}_i\) and the plant TPN\(^{plant}_i\), 2) interaction between LCs TPN\(^{ctrl}_i\) and communication channel(s) TPN\(^{channel}_i\), and 3) runtime environment changes based on issued commands (e.g., variable updates, execution delays).

Thus, we introduce methods for automatic extraction of TPN-based controller models from existing CIPN-based models that may be used to generate control code. We capture interaction between the automation system and plant, as well as the platforms' runtime environment in Section IV-A, and security-aware modeling of communication channels and their interfaces with controllers in Section V. These methods enable developing a complete closed-loop system model that can then be used to reason about system resiliency to attacks.

A. Modeling Plants and Controller-Plant Interaction

Nominal behavior of the controlled physical system is typically known at control design time. Thus, since the formalism of CIPNs is universally adopted for automation design, we assume that a PN-based (i.e., CIPN or TPN) plant model is available.\(^3\) On the running example, we describe development of such TPN-based plant model, along with a TPN-compliant controller-plant interface implemented through marking-dependent guard functions.

\(^2\)This captures the general case where time or frequency multiplexing may be used to provide multiple communication channels over the same medium.

\(^3\)On the other hand, if an automata or other discrete-event system representation of the plant is available, existing tools and methodologies can be used to translate such models into a PN-based representation (e.g., [38], [39]).

Fig. 3(a) shows a TPN model of the pick & place station from Fig. 2; Fig. 3(c) models the incoming workpieces arrivals, with a lower bound on the interarrival times. Place Pp&p_Init represents the station’s initial state. Token flow from this place is conditioned by the corresponding commands of LC\(_i\) PP_Act==1 and PP_Act==2 from the controller model in Fig. 2(c). In the formalism of TPN, marking-dependent guard functions can be used to restrict state changes (i.e., token flow) in the plant model, i.e., a marking-dependent function (denoted with \( M(\cdot) \)) assesses the state of the controller model (i.e., token distribution) and returns the current number of tokens inside the argument place. Hence, guard function \( G:M(\text{Pctrl}_P&P1)==1 \) (or \( G:M(\text{Pctrl}_P&P2)==1 \) for the other conveyor belt) is associated with transition Pp&p_wfPick1 (or Pp&p_wfPick2). Once the pick & place process is triggered (i.e., LC\(_i\) initiates it from a specific conveyor belt) by the LC\(_i\)'s model advancing its token to Pctrl_P&P1 (or Pctrl_P&P2), the station’s token transitions to place Pp&p_P&P1 (or Pp&p_P&P2) if conveyor belt 1 (or 2) has a workpiece waiting to be processed. Note that in order to capture realistic executions, the actual times to complete the pick & place and return processes are not deterministic; this is natively supported by TPN’s timed transitions; transitions Pp&p_P&P1/Tp&p_P&P2 have firing times from the interval \([\text{proc}_{min},\text{proc}_{max}]\), as shown in Fig. 3(a).\(^4\)

\(^4\)These intervals may be obtained experimentally under nominal conditions at runtime, or based on design constraints at design-time. Also, different durations of the picking process from different conveyors are supported but set equal in the above model/figure to simplify our presentation.
Now, the station dwells in place $P_{p&p\_wfRet}$ waiting for the signal from $LC_2$ to return to the home position, i.e., guard function $G:\text{M}(P_{ctrl\_Ret})==1$ conditions transition $T_{p&p\_wfRet}$ on the corresponding controller’s state (i.e., where the return action is issued). Once return is commanded, the pick & place station takes non-deterministic time from $t_{ret}\_p&p\_lkp$ to return (i.e., transition over $T_{p&p\_RetInit}$). After this transition, the station model is back in the initial state, waiting to process the next workpiece.

The actuation part of the plant-controller interface is managed by guard functions assessing the controller’s marking; therefore, explicit actuation input updates (e.g., $PP\_Act=1$) in the CIPN places are omitted in the transformation to the TPN model as TPN places do not feature any attributes. This interface can also be achieved alternatively, by utilizing update functions which are triggered on the firing of controller’s transitions and can update markings or variables. The choice of the transformation semantics from CIPN to TPN can therefore be adjusted to the specific platform implementation.

Dually to the actuation part of the interaction, sensing is modeled by introducing plant-marking-dependent guard functions on controller’s transitions. Specifically, transitions conditioned by sensor values in the form $\text{sensor==value}$ in the CIPN controller model are replaced with immediate transitions guarded by a Boolean function evaluating to $true$ if the plant model marking corresponds to the plant state where $\text{sensor==value}$ is satisfied, and to $false$ otherwise.

For instance, once $LC_2$ commands return of the pick & place station, (i.e., controller model from Fig. 2(c) has the token in $P_{ctrl\_Ret}$), it is blocked on the transition $T_{ctrl\_wfRet}$ which is guarded by the condition $\text{Ret\_Complete==1}$. This transition in the CIPN model is transformed into a transition in the TPN model in Fig. 3(b) that is guarded by a function dependent on the marking of the pick & place station model from Fig. 3(a) (i.e., controller waits for the station to reach home position). Guard function $G:\text{M}(P_{p&p\_Init})==1$ returns $true$ when the token in the plant model transitions from $P_{p&p\_Ret}$ to $P_{p&p\_Init}$ over the timed transition $T_{p&p\_RetInit}$ (here, $\text{M}(\cdot)$ denotes a function that returns the current number of tokens inside the argument place) — hence, controller $LC_2$ can transition over $T_{ctrl\_wfRet}$. Therefore, this guard function is used for the transition $T_{ctrl\_wfRet}$ in the TPN model of $LC_2$. This models the controller side of the controller-plant interaction, i.e., sensor sampling. More complex conditions based on multiple sensors are implemented by forming an arbitrary plant marking-dependent Boolean guard function.

Fig. 3(b) shows a controller model of the described controller-plant interface. The model, obtained from the CIPN-based model in Fig. 2(c), is intermediary, and not fully TPN-compliant; the CIPN-based communication semantics (i.e., signal transmissions via $\text{Send}(\text{destination},\text{signal})$, and receptions through $\text{signal==value}$ denoted in red in Fig. 3(b)) is still present in the model. However, to allow for verification of properties for systems where networking is not a concern, this communication semantics can be easily adapted to TPNs by applying the same $\text{guard/update}$ functions as described; this results in a model architecture from Fig. 4(a).

Additionally, note that the conveyor monitor model from Fig. 2(b) can be directly transformed into a TPN, with guards $\text{Pres1==1}$ and $\text{Pres2==1}$ conditioning progress based on external inputs (i.e., the process) triggered by the process model shown in Fig. 3(c). This process model abstracts away the nature of the process of incoming workpieces on the conveyors with a minimum inter-arrival time (i.e., time in the interval $[t_{\text{proc}},\infty]$).

1) Controller Runtime Environment Modeling: Another challenge for the automatic mapping of CIPN-based control models into TPN-compliant models is mapping of places issuing system calls from the runtime environment (e.g., execution delays, requests for timer interrupts, setting counter events) or updating local controller state (e.g., manipulating global variables). Requested execution delays can easily be modeled as timed transitions with the exact firing times (i.e., where the lower and upper firing time bound are the same); in general, however, event timings with different semantics are available depending on the control implementation—i.e., GRAFCET or SFC. In [36], [37], authors provide detailed translational semantics between CIPNs and TPNs in these cases by introducing event sequencers as certain conditions exist where transitions can be taken while time to some events generated internally in places has still not elapsed.

Remark 2 (Modeling more Complex Execution Environments): While we consider single-threaded automation examples (as most sequential control implementations are), existing techniques for modeling parallel systems can be applied given the expressiveness of TPNs. For example, for multithreaded applications where task preemption is allowed, the operating system scheduler can be modeled as a separate component, even in case of multi-processor platforms [43].

$\square$

B. CIPN and TPN Controller Equivalence

An execution path in CIPNs can be defined as a sequence of markings, where a change in the marking occurs due to firing of a transition. Recall that places are associated with actions; hence, each marking is associated with a set of actions, while transitions are associated with guards—firing of each transition is thus conditioned by a set of conditions. Thereby, an execution path is a sequence $M_0, T_1, M_1, T_2, ...,$, where $T_i$ is the transition taking the net from marking $M_i$ to $M_{i+1}$. In the TPN model, a path is characterized by a similar sequence with the addition of transition timing. In our case, it is sufficient to maintain the CIPN controller execution paths in the TPN model, as our objective is operational equivalence of the source (centralized), and the target (distributed) control models.

CIPN controller specifications are fully deterministic by design, and have only immediate transitions. Thus, the corresponding target TPN-based controller models obtained by directly constructing the same model in the TPN formalism without additional constructs (i.e., as previously described in

$^5$We employ the standard assumption that all inputs are re-evaluated after firing of every transition (e.g., as done in [37]).

$^6$Strictly speaking, two types of time intervals characterize each transition: static intervals (i.e., design-time bounds) when they may fire, and dynamic (i.e., runtime) intervals when they can fire at any given instant, conditioned by all other enabled transitions. However, for purposes of showing marking-based equivalence with CIPNs, time can be abstracted away.

$^7$Immediate transitions become fireable immediately after enabling, i.e., without a time delay.
We assume a powerful network-based adversary that:
1) Has the full knowledge of the distributed system, including the CIPN models, generated code and analysis framework, as well as the current state of all LCs (and their transceivers).
2) Has network access and full communication protocol compliance, i.e., is able to transmit unsigned messages as any of the LCs, or intercept messages or ACKs exchanged by LCs.
3) Is able to precisely time actions and align transmissions with legitimate network traffic, e.g., to interfere with legitimate messages by transmitting the carrier signal or a protocol-compliant message.

Thus, the adversary may mount the following attacks:
1) *Interception or delaying of legitimate packets (DoS)*: With these attacks, adversarial transmissions occupy the channel (a) blocking transmissions from legitimate LCs to prevent or delay their access to the network, or (b) blocking ACKs on legitimate LC’s transmissions to cause unnecessary retransmissions and slow down progression of the targeted transmitter [10].
2) *ACK spoofing*: The attacker may impersonate an ACK expected by a legitimate transmitter; e.g., following by interception of the transmission, the attacker may spoof the ACK misleading the legitimate transmitter into believing that the transmitted signal was received by the intended receiver [9], both for regular and ‘heartbeat/sync messages [44].
3) *Impersonation/Masquerade*: The adversary may transmit false event signals on behalf of a legitimate LC (i.e., impersonating another controller), with the goal to inject false commands [9] or sensor measurements; such attacks could e.g., allow the targeted receiver to resume execution while it is blocked waiting for an event signal, before the event it is sent by the legitimate LC.
4) *Signal replacing/Message modification*: The adversary may modify content of a legitimate message to deliver false event information. While logically the same, the attack procedure differs from intercepting a legitimate transmission followed by a masquerading attack [11], and thus is modeled differently.\(^8\)
5) *Replay attack*: The attack characterizes an adversary that records events signaled by the LCs and replays the sequence of events on behalf of one or more LCs; thus, maliciously emulating activity of LCs whose operation he/she is interfering with [11].

The above attack set, considered in this work, covers all reported attacks that, from the standpoint of low-level signaling of events, could have direct impact on Quality-of-Control (QoC) of the underlying physical process [7], [45]. Other attacks, such as attacks on network routing policies, are focused on higher-level information flows and are thus harder to directly relate to automation QoC [11]. Consequently, our goal is to model attacks by capturing their influence on the sequential control system and the resulting QoC, rather than the employed attack vector for any specific attacks; i.e., the attack model should be agnostic to the actual attack implementation.

**B. TPN-Based Modeling of Attack Impact**

Recall that CIPN models rely on platform-provided communication APIs for passing events between LCs; e.g., as in Fig. 5(a), Send(destination, signal=value) command within a place sends the updated value of signal to the destination LC, while condition signal==value on a transition within the model blocks execution until the signal corresponding to the desired value is received over the network. To enable formal analysis of the attack impact on QoC of distributed automation, it is necessary to develop a TPN-compliant model of the interface (i.e., transceiver) between the controller and security-aware channel model; such model can be then composed with the TPN-based models described in Section IV-A, resulting in Fig. 4(b) architecture.

Such security-aware formal model has to capture:
1) *Application-level* (i.e., controller side) communication stack behavior, directly affected by (2) the channel-side (i.e., communication medium) attack model, and (3) the controller-channel interface. Specifically, application-level (i.e., control-related) communication stack behavior, such as delays or

\(^8\) This type of attack is technically more challenging to perform compared to other attacks, especially over a wireless medium.
blocking on communication peripheral resources, is of interest for security analysis, as this presents the main reflection of the communication-level attacks onto the control functionality. Therefore, when translating the CIPN communication model from Fig. 5(a) into a TPN-compliant model, it is necessary to capture application software states that directly affect progress of the control functionality, conditioned by data dependencies resolved via communication. Such models can be obtained from the actual application firmware running on the embedded LCs (i.e., source code). For example, when IEEE 802.15.4 protocol is used, as in the case study presented in Sec VII, the state-machine/TPN representation can be directly extracted from the radio driver (as done in Fig. 5(b)). On the other hand, if more complex communication stack is considered (i.e., implementing higher communication layers), exiting state-machine extraction techniques (e.g., [20]) can be employed.

Second, the channel model has to explicitly capture the channel states essential for supporting the attack models presented in Section V-A—channel states that are not observable (or alterable) need not be modeled (e.g., bit-level signaling, or carrier-level modulation). Finally, a TPN-compliant interface between the controller and security-aware channel models is needed to allow for their formal composition, enabling system-level analysis of adversarial influence on the entire system. Therein, specific data link layer (OSI model layer 2) features are crucial for understanding retransmissions and ACK mechanics which, as we will show, affects design of attack detectors. Therefore, while controller models should capture application-level communication semantics, it is also necessary to include protocol-level details within the transceiver (XCVR) models, which act as the interface between the controllers and the medium (as shown in Fig. 4(b)). XCVR specifics are commonly available for the specific employed radio communication chip as RF circuitry control is usually state-machine based (e.g., referred to as the internal RF state machine [46] in the case of radios used in our implementation).

On the other hand, explicit security-aware channel modeling is medium-, protocol-, and attack-dependent. Fig. 5(c) and Fig. 6 show a security-aware model of a half-duplex, acknowledge-required unicast CSMA-CA-based communication channel with respect to the previously defined attack model. While other medium/protocol variants can be easily modeled due to the expressiveness of TPNs, we consider this specific model as it applies to our physical setup described later in Section VII. In the remaining of this section, we describe the transformation from the CIPN-based LC communication model to a TPN-compliant model assuming the aforementioned channel, while aiming to balance between the model expressiveness and capturing security-aware behavior required for analysis of QoC under attack.

C. Security-Aware Modeling of the Channel and Controller-Channel Interaction

Fig. 5(b) shows the TPN transmitter/receiver models that replace the platform-independent CIPN transmitter/receiver model in Fig. 5(a). Fig. 5(c) shows the nominal channel model (i.e., without adversarial influences), while Fig. 5(d) shows the transceiver (XCVR) model. Notice that both $LC_A$ and $LC_B$ have identical transceivers; thus, $N \in \{A, B\}$ in place/transition names. Table I enumerates symbols (local flags, variables, and transition timing parameters) used in the models in Figs. 5, 6. The internal RF state machine can be in the listening, transmitting a packet, waiting for acknowledgement, or transmitting an acknowledgement states. The transceiver employed in our case study (in Sec. VII), performs up to three
retransmissions before signaling a transmission failure to the application. On the application level, an unbounded number of retransmissions are performed in case the transceiver returns failure. The TPN model in Fig. 5(b-d) models this interaction.

In the remaining of this section, we show how the attacks described in Section V can be modeled as TPNs. Specifically, we describe additional places, transition, and arcs to be added to the nominal channel model shown in Fig. 5(c) to capture the attacks. To enhance model readability, Fig. 6 depicts only additional places and transitions in red color required to model a specific attack, while the nominal places and transitions are depicted in black (all parts of the nominal model not relevant for the specific attack are omitted therein).

a) DoS attack submodel: Fig. 6(a) shows the DoS attack submodel. When the channel is idle, the attacker may decide to occupy the channel to prevent legitimate transmissions. He/she may do so at any time (non-deterministic choice) when the channel is not busy, and keep the channel busy arbitrarily long. In the model, the channel is kept busy for some non-deterministic time in the range $[\Delta_{Ch}, \Delta_{DoS}]$, after which it is released by the attacker.

b) ACK interception/spoofing submodel: Fig. 6(b) shows the ACK intercept/spoof submodel. To model ACK interception, an additional transition is needed allowing the channel to return to idle state following ACK transmission, without the transmitter ($LC_A$) receiving the ACK sent by the receiver ($LC_B$); i.e., transition $Tch_{TxAckInt}$ is added as shown in Fig. 6(b), and is not associated with the update function $U: BXCVR_{RxAck}=1$. However, when ACK spoofing is considered, the attacker may transmit an ACK when the targeted receiver is not in the process of acknowledging, while the targeted transmitter is in the process of waiting an ACK.

Additionally, malicious ACK spoofing may be performed when the signal to which the ACK is intended to correspond is transmitted already, but the ACK has not yet been received by the sender (e.g., due to an intercepted ACK from the legitimate receiver). This is enabled with the additional net branch in Fig. 6(b) starting with transition $Tch_{wfAckImp}$. As a result of firing of this transition, the channel is declared busy and the spoofed ACK is assumed to take the same time as transmitting legitimate ACKs; thus, the transition $Tch_{TxAckImp}$ has the same attributes as $Tch_{TxAck}$, with the exception of the signal to the targeted transmitter signalling ACK transmission is done (i.e., update $U: AXCVR_{RxAck}=1$ is omitted).

c) Message intercept/modify submodel: Fig. 6(c) shows the message intercept/modify submodel, where an additional transition $Tch_{TxMsgMod}$ ($Tch_{TxMsgInt}$), represented as one transition for conciseness, is added. In the case of the modification attack, this transition in the model allows the attacker to deliver a signal different form the one originally transmitted (i.e., $U: BXCVR_{PRx}=$ $\text{Fmod}$ where $\text{Fmod}$ is the payload modified by the attacker). In the case of packet interception, no update to the receiver’s XCVR receiver buffer is made, and consequently the XCVR is not notified of a received packet (i.e., update functions are omitted and denoted as $(.)$ in Fig. 6(c)).

d) Message impersonation submodel: Fig. 6(d) presents the masquerade submodel. The additional transitions and places allow the attacker to make a non-deterministic choice to impersonate transmission of the expected transmitter whenever the channel is not busy, the targeted receiver is waiting for the corresponding signal, and the original transmitter is not in the process of sending this signal. Then, similarly to the nominal (legitimate) transmission model (shown in Fig. 5(c)), the transmission takes a non-deterministic time in the same range as legitimate transmissions. Note that the received payload on $LC_B$ is in this case the value $\text{Pinj}$ crafted by the attacker, rather than $AXCVR_{PTx}$, normally transmitted by $LC_A$ in the adversary-free case.

Remark 3 (Replay attacks): Due to the introduced non-determinism, any specific sequence of attack actions are contained within the presented model (as long as the individual actions correspond to the attack model from Section V). Thus, replay attacks are covered by the presented model as they are only specific executions of the presented security-aware channel model. On the other hand, using a similar approach, finite memory replay attacks can be captured by a model that restricts inserted attack signals only to the previously transmitted messages, as done in [7].

Remark 4 (Controller-plant VS. controller-channel interface modeling fidelity): Control interface to the channel is modeled in far more detail than the interface to the plant, by abstracting away locally-connected actuator drives, relays, analog amplifiers, etc. The reason is that, in this work, we do not consider physical plant-level attacks. Hence, modeling the controller-plant interaction at a lower level of abstraction would unnecessarily increase model complexity. However, the presented techniques can be easily extended and the framework fully adapted to also cover physical attacks on the plant. □

VI. RESILIENCY ANALYSIS AND SECURITY PATCHING

A security-aware closed-loop system model obtained by composing the developed security-aware TPN models can be used to verify system-level safety and QoC properties in the presence of attacks. TPN analysis tools (e.g., [16],

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>SW acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChBusy</td>
<td>Indicator whether the channel is currently busy</td>
<td>YES</td>
</tr>
<tr>
<td>N_RxBuf</td>
<td>Local RX buffer</td>
<td>YES</td>
</tr>
<tr>
<td>N_RxAck</td>
<td>Local flag indicating successful transmission, i.e., ACK reception</td>
<td>YES</td>
</tr>
<tr>
<td>NXCVR_PTx</td>
<td>Transceiver Tx payload buffer</td>
<td>YES</td>
</tr>
<tr>
<td>NXCVR_Rx</td>
<td>Transceiver Rx payload buffer</td>
<td>YES</td>
</tr>
<tr>
<td>NXCVR_TX</td>
<td>Signal to XCVR initiating transmission</td>
<td>YES</td>
</tr>
<tr>
<td>NXCVR_Rx</td>
<td>Signal to XCVR indicating transmission</td>
<td>NO</td>
</tr>
<tr>
<td>NXCVR_TxAck</td>
<td>Signal from XCVR indicating reception</td>
<td>YES</td>
</tr>
<tr>
<td>NXCVR_RxAck</td>
<td>XCVR signal indicating ACK transmission</td>
<td>NO</td>
</tr>
<tr>
<td>NXCVR_RxCh</td>
<td>Signal to XCVR initiating ACK transmission</td>
<td>NO</td>
</tr>
<tr>
<td>NXCVR_RxCh</td>
<td>Signal from XCVR indicating reception</td>
<td>NO</td>
</tr>
<tr>
<td>NXCVR_RxCh</td>
<td>Transceiver Rx payload buffer</td>
<td>YES</td>
</tr>
<tr>
<td>NXCVR_RxCh</td>
<td>Message transmission time (bounds)</td>
<td>—</td>
</tr>
<tr>
<td>Tch_TxAck</td>
<td>ACK transmission time (bounds)</td>
<td>—</td>
</tr>
<tr>
<td>Tch_TxAck</td>
<td>ACK transmission time (bounds)</td>
<td>—</td>
</tr>
<tr>
<td>Tch_TxAckImp</td>
<td>Back-off time (bounds)</td>
<td>—</td>
</tr>
<tr>
<td>Tch_TxAckImp</td>
<td>Data link layer ACK timeout</td>
<td>—</td>
</tr>
<tr>
<td>Tch_TxAckImp</td>
<td>Application-level ACK timeout</td>
<td>—</td>
</tr>
<tr>
<td>Tch_TxAck</td>
<td>Contention time due to DoS (bounds)</td>
<td>—</td>
</tr>
</tbody>
</table>

SYMBOLS USED IN FIG. 5 AND FIG. 6; THIRD COLUMN (WHERE APPLICABLE) INDICATES ACCESSIBILITY TO APPLICATION SOFTWARE (OR ONLY TO THE TRANSEIVER’S INTERNAL RF STATE MACHINE).
Fig. 6. Additional places, transitions, and arcs required to obtain a security-aware channel model for different attack types: (a) DoS, (b) ACK intercept/spoof, (c) message modification, and (d) masquerade. All attack-related components of the model are depicted in red color, while nominal components are shown partially for completeness in black (where relevant).

![Diagram](image_url)

[47] allow for verification of formal properties specified as Linear Temporal Logic (LTL), Computational Tree Logic (CTL), or Timed CTL (TCTL) formulas [48]. In this work, we employ the tool Romeo [16] that enables verification of TCTL-based formal queries, such as traditional safety (e.g., absence of deadlock). In addition, as plant models are included, we can specify relevant domain-related plant-state-bound properties that are crucial for functional safety and QoC assessment. For our running example, the considered properties include:

**Property 1:** A workpiece on conveyor 1 never triggers a pick-up from conveyor 2: this can be formally captured as: \( AG(\neg (M(Pp\&P_P\&P2)==1 \land \text{activeConveyor==1})) \), where A and G are quantifiers signifying formula satisfaction along all paths and always (i.e., along all subsequent paths), respectively.

**Property 2:** A workpiece detected on any of the conveyors is eventually picked-up from conveyor 2; this can be formally captured as: \( AG(p \Rightarrow AF(q)) \), where \( p \Rightarrow q \) means that for all executions, continuous satisfaction of property \( p \) implies always eventual satisfaction of property \( q \), or formally \( AG(p \Rightarrow AF(q)) \).

**Property 3:** The pick & place station does not commence cycle (i.e., it is neither in \( Pp\&P_P\&P1 \) nor \( Pp\&P_P\&P2 \)), while the conveyor monitor is waiting for incoming workpieces (i.e., in the place \( Pcm\_Init \)). Formally, \( AG(M(Pcm\_Init)\land M(Pp\&P_P\&P1)\land M(Pp\&P_P\&P2)==0) \).

Using the Romeo tool, we verified that these as well as other QoC- and safety-critical properties are not satisfied in the presence of attacks, as the attacker is capable of significantly altering the intended interaction between LCs, at arbitrary moments in time. For example, Property 1 is violated under *message modification* attacks, Property 2 under possible infinite *DoS*, while Property 3 fails under *spoofing*. The aforementioned properties are violated regardless of the values of timing parameters used in the model. Note that bounds on time-to-transmit and time-to-acknowledge can be obtained from experimental measurements, or directly from network specifications. Also, transceiver-related timings (e.g., back-off time during clear channel assessment) can be obtained from the employed transceivers’ specifications.

Regarding verification scalability—in the system model, one nominal pick & place cycle, with all attacks disabled, contains around 35 transitions, which is on the order of the number of states in the model. Model complexity increases with the addition of non-deterministic attack choices, besides the time-induced non-determinism (in the plant model). Yet, in all cases, the tool takes less than 1 s to find an execution path violating the properties, on a workstation with an Intel i7-8086K CPU (4 GHz clock) and 64 GB memory.

**A. Addressing the Discovered Vulnerabilities**

As our previous analysis have shown, attack actions may significantly affect performance of distributed IoT-based industrial automation systems; to address them, it is necessary to add certain security mechanisms. In this section, we discuss how such security mechanisms affect system models and verifiability of the relevant properties.

1) **Detecting Denial-of-Service Attacks:** Packet and acknowledgement (ACK) drops are common in wireless communication, and hence ACK and retransmission mechanisms are commonly used in such setups. For instance, in our experimental setup described in Sectio VII, ACK request can be disabled in transceiver settings, in which case no
retransmissions are attempted on the data link layer. For two isolated transceivers, this amounts to the one-way packet success rate of approximately 99\% (see histograms in Fig. 8 that exclude unsuccessful transmissions). Thus, when ACK requests are enabled, up to three data-link layer retransmissions are performed, and we experimentally observed that no application-level retries are required beyond the three low-level protocol-provided retransmissions, in the case when a single industrial machine operates in isolation. 

On the other hand, to increase network utilization, we emulated a number of additional machines communicating over the same wireless channel in physical vicinity (described in more detail in Section VII); we experimentally observed the one-way packet success rate of approximately 98\%. Thus, two application-level retries were sufficient to enable reliable exchange of events, ensuring correct operation. Intuitively, protocol-provided retries are issued in short bursts while application-level retransmissions incur significant delay; the channel is more likely to be continuously busy for a short period of time (e.g., occupied by other legitimate transmissions). Yet, an adversary may repeatedly deny network access to legitimate controllers preventing the system from progressing. Consequently, the modeled system does not satisfy Property 2, despite application-level retransmissions, unless DoS attacks can be detected and system halted (or other precautionary actions taken), using e.g., a separate secure channel.

From the operational perspective, every LC may implement a limited number of successive application-level retransmissions before declaring that it is under attack. For instance, if in our setup from Section VII, we limited the number of retransmissions to five, amounting to a theoretical one-way packet success rate of eight nines (if application-level retransmissions are assumed to be independent). To address this from the modeling perspective, we add an additional place where the transmitter’s model transitions to, when application-level retries are exhausted (see Fig. 7(a)). Hence, we can verify that if infinite blocking of medium access is allowed, LCs may end up in the place Pa_DoSDetect. Conversely, if DoS attacks are limited to four consecutive channel access denials, Property 2 is satisfied. Note that immediate emergency halt of the machinery may not be possible if a secure communication channel is not available or the DoS attacks cannot be isolated from the network (e.g., using bus guardians).

2) Authentiﬁng network ﬂows: Traditional cryptographic techniques for ensuring integrity of network ﬂows rely on signing packets with Message Authentication Codes (MAC) [9], and can be used to defend against spoofing attacks. In this setting, every transmission between LCs is signed by the transmitter using a secret key, and the signature is verified by the receiver; therefore, the attacker cannot tamper with the message payload, or else he/she will be detected.

From the modeling perspective, introducing authentication can be modeled as an additional condition on the receiving transitions (in the controller models) where the received payload is compared to desired values; i.e., the MAC portion of the payload is compared to a secret value that cannot be altered (in the case of modiﬁcation) or generated (in the case of spoofing attacks) by the attacker. Speciﬁcally, the transition \(T_b_{\text{wfRx}}\) in the \(LC_{b}\) model in Fig. 5(b) would feature an additional guard function on the \(B_{\text{RxMAC}}\) variable. 

Optionally, if the signature veriﬁcation fails, a transition to a place modeling intrusion detection reaction can be added as shown in Fig. 7(b); this is left to the application designer as reacting to detected intrusions is highly application-speciﬁc.

Using the developed framework, we veriﬁed that if non-authenticated transmissions are not allowed (i.e., authentication implemented), Properties 1 and 3 can be verified over our running example, under the condition that inﬁnite denial of network access to LCs is not allowed, as previously discussed.

3) Acknowledgement Spoofing: Authenticating transmission does not affect ACKs as the data-link layer is responsible for ACK packets while MACs are added to the packet payload. In addition, non-encrypted sequence numbers, which are part of the packet frame, can be overhead by the attacker. Thus, valid ACKs can be generated on behalf of inactive (failed) LCs. Also, undelivered (i.e., intercepted) transmissions can be falsely acknowledged, even when authentication is used. This is a well-known shortcoming of data link layer ACKs [11], [49], and could be alleviated by application-level ACKs. Enforcing consensus over event-propagation in discrete event systems spans beyond the scope of this paper; yet, the presented modeling techniques can be utilized to model additional implemented protocols.

While this section introduced the general security-aware modeling aspects, with occasional focus on speciﬁc medium access techniques to avoid overly general discussions, in the following section we demonstrate the use of the presented framework on real-world industrial case studies.

VII. CASE STUDIES: INDUSTRIAL MANIPULATORS

We consider a full physical implementation of a reconﬁgurable industrial pneumatic manipulator with a variable number of modules/degrees of freedom (DOF) controlled in a distributed fashion; i.e., one local controller per module/DOF. We demonstrate effectiveness of our framework on multiple module conﬁgurations (i.e., 2-DOF, 3-DOF).

A. 2-DOF Industrial Pneumatic Manipulator

The pneumatic industrial manipulator in the 2-DOF conﬁguration is depicted in Fig. 9(a); two double-acting cylinders (denoted \(A\) and \(B\)) provide translational degrees of freedom, while the pneumatic gripper (denoted \(C\)) provides means of handling the workpiece. All actuation commands are issued by updating electrical signals \(\times_p, x \in \{a, b, c\}\) which activate

\[\text{Fig. 8. Tx-to-Rx and Rx-to-Ack times measured on our IEEE 802.15.4-enabled LC platform described in Section VI.}\]
monostable dual control pneumatic valves. Notice that signals are denoted with \( x \) while cylinders are denoted with \( X \). Cylinders \( A \) and \( B \) are equipped with two proximity switches which allow position (i.e., fully retracted, fully extended) sensing. Signals corresponding to fully retracted (home) position are denoted \( x0 \), while fully extended (end) position signals are denoted \( x1 \). Additionally, the system contains a start switch whose corresponding signal is denoted by \( st \).

CIPN-based models of three LCs are shown in Fig. 10. Initially, cylinders \( A \) and \( B \) are fully retracted, and gripper \( C \) released—in this state the manipulator is ready to begin its work cycle. The initial work cycle of the manipulator is started by pressing the start switch \( (st = 1) \), after which operation is fully automated. First, cylinder \( B \) extends towards the workpiece picking position (due to actuation command \( b0 = 1 \)). Once cylinder \( B \) reaches its end position \( (b1 = 1) \), gripper \( C \) is commanded gripping \( (cp = 1) \). Controller \( B \) waits for 500 ms for the part to be gripped. Then, cylinder \( B \) retracts (due to command \( b0 = 0 \)), and once it reaches home position \( (b0 = 1) \), cylinder \( A \) extends (due to command \( ap = 1 \)). After reaching its end position \( (a1 = 1) \), cylinder \( B \) extends towards the placing position (due to command \( b0 = 1 \)). Once it reaches its end position \( (b1 = 1) \), gripper \( C \) is commanded released of the workpiece (command \( cp = 0 \)). 500 ms later, cylinder \( B \) retracts \( (b0 = 0 \text{ followed by } b0 = 1) \), after which cylinder \( A \) retracts \( (ap = 0 \text{ followed by } a0 = 1) \). The manipulator returns into its initial state, after which the next cycle is automatically executed. Signals (i.e., sensors outputs and actuator inputs) are allocated to LCs according to their physical proximity: \( \{ap, a0, a1\} \) are mapped to controller A (i.e., \( LC_A \)), \( \{bp, b0, b1, st\} \) to controller B (\( LC_B \)), and \( \{cp\} \) to controller C (\( LC_C \)).

TPN models are obtained from these specifications as described in Section IV, but are omitted here due to their size. On the other hand, pneumatic cylinders are modeled as two-state plants with bounded, non-deterministic extending/retracting times obtained from experimental measurements. We extract timing parameters (i.e., bounds on time-to-transmit, time-to-acknowledge, and back-off timing) from experimental measurements—histograms for 10,000 messages are shown in Fig. 8, for the employed low-cost ARM Cortex-M3-based controllers equipped with an IEEE 802.15.4-compliant transceiver. On the other hand, we obtain transceiver-related timings (e.g., back-off time during clear channel assessment) from the radio specifications [46]. While we verified a large number of safety and liveness properties for this setup, we illustrate verification and security patching on a more complex 3-DOF setup.

B. 3-DOF Industrial Pneumatic Manipulator

A 3-DOF configuration of the described manipulator is shown in Fig. 9(b). The additional rotational DOF, provided by cylinder \( C \), introduces an additional LC and increases the complexity of the LC coordination. This configuration may be used to prepare workpieces for painting by immersing them into a pool with cleaning solution, and returning them to the pick-up position for further processing by another machine. Fig. 9(c,1) shows the physical setup for this configuration; the upper portion of the manipulator is shown such that cylinders are visible. Fig. 9(c,2) shows the low-cost ARM
Cortex-M3-based LC with the corresponding IEEE 802.15.4 transceiver. While the models are more complex than in the 2-DOF case, they are semantically similar and thus omitted. Fig. 11(a) shows event timing—i.e., states of all sensing and actuation signals, for one sample pick-immerseshake-return run; messages exchanged by LCs are denoted with blue arrows. X axis is unlabeled as the speed of the workcycle can be controlled by regulating air pressure in the system and is thus not crucial.

Property 4: Gripper D is always gripped before cylinder B picks the workpiece; formally captured as, $\text{AG}(\text{M(PdGrip\_Gripped)}==1 \land \text{M(PbCTRL\_bCyl\_Extend1)}==1)$. 

Property 5: A workpiece is eventually pro-processed, once the work cycle is started. Formally, $\text{M(PbCTRL\_bCyl\_Extend1)}==1 \rightarrow (\text{M(PcCTRL\_cGrip\_Release)}==1)$. 

When no security mechanisms are employed, we verified violation of these properties. Property 4 is violated due to a possible impersonation attack at the gripper controller; an attacker may send the command to release the workpiece before it was returned to the return position. Fig. 11(b) shows signal timings acquired on a sample cycle run in which the workpiece is dropped due to a maliciously injected command to release the gripper (potentially causing mechanical damage to the workpiece and/or the manipulator).

However, if transmissions are authenticated, and the model adjusted correspondingly as described in Section VI-A2, this vulnerability is alleviated. We applied a software security patch by including the mbed TLS (Secure Sockets Layer) library that our IoT controllers are fully compatible with. Signing a 128 bit message authentication code over one transmitted signal incurs computational overhead of $\sim 100 \mu s$ on the employed low-cost ARM Cortex-M3-based LCs; this practically negligibly slows down manipulator’s work cycle, while providing security guarantees. Hence, Property 4 is satisfied following this security patch.

Property 5 is violated due to the possibility of a DoS attack that infinitely delays progress. From the model’s perspective, this attack does not cause a deadlock—while the physical process is stalled, the cyber process is in fact livelocked, reattempting to access the channel (i.e., same places are revisited and same transitions fire infinitely often). Fig. 11(c) shows signal timings acquired on a sample run where a DoS attack is launched by enabling carrier transmission on the attacker’s transceiver, in order to jam messages after the workpiece was picked up from the immersion pool. As described in Section VI-A1, wireless control nodes can keep track of unsuccessful medium access attempts, and promptly halt operation when a DoS attack is detected. In such cases, distributing the information about DoS detection requires a secure channel, which we do not consider in this work.

### VIII. CONCLUSION

In this paper, we have developed a framework for security analysis of distributed sequential control systems captured as widely-adopted CIPN-based models. As CIPNs do not support verification of formal safety properties in the presence of attacks, we transform controller models into TPNs that inherently enable this verification by supporting non-deterministic timed transition as well as non-deterministic choice among transitions; this is crucial as it imposes minimal assumptions on adversarial actions. We have shown how a model of a network-based attacker can be integrated into the non-deterministic communication channel model, and have verified violation of safety properties in presence of attacks.

Additionally, we have shown how results of verification can be used to pinpoint vulnerabilities in the control software implementation, and suggest security patches to alleviate impact of these vulnerabilities on control performance; we have also provided the loop back to the modeling stage enabling verification of the same safety properties that are now satisfied due to the use of appropriate security mechanisms. Finally, we have evaluated our framework on an industrial case study of a realistic scale and complexity.

### ACKNOWLEDGMENT

This work is sponsored in part by the ONR under agreements N00014-17-1-2012 and N00014-17-1-2504, as well as the NSF CNS-1652544 grant. It was also partially supported...
by Serbian Ministry of Education, Science and Technological Development, research grants TR35004 and TR35020.

REFERENCES


