Reliable Industrial IoT-Based Distributed Automation

Vuk Lesi
Duke University
Durham, North Carolina
vuk.lesi@duke.edu

Zivana Jakovljevic
University of Belgrade
Belgrade, Serbia
zjakovljevic@mas.bg.ac.rs

Miroslav Pajic
Duke University
Durham, North Carolina
miroslav.pajic@duke.edu

ABSTRACT

Reconfigurable manufacturing systems supported by Industrial Internet-of-Things (IIoT) are modular and easily integrable, promoting efficient system/component reconfigurations with minimal downtime. Industrial systems are commonly based on sequential controllers described with Control Interpreted Petri Nets (CIPNs). Existing design methodologies to distribute centralized automation/control tasks focus on maintaining functional properties of the system during the process, while disregarding failures that may occur during execution (e.g., communication packet drops, sensing or actuation failures). Consequently, in this work, we provide a missing link for reliable IIoT-based distributed automation. We introduce a method to transform distributed control models based on CIPNs into Stochastic Reward Nets that enable integration of realistic fault models (e.g., probabilistic link models). We show how to specify desired system properties to enable verification under the adopted communication/fault models, both at design- and run-time; we also show feasibility of runtime verification *on the edge*, with a continuously updated system model. Our approach is used on real industrial systems, resulting in modifications of local controllers to guarantee reliable system operation in realistic IIoT environments.

CCS CONCEPTS

• **General and reference → Cross-computing tools and techniques**; • **Computer systems organization → Dependable and fault-tolerant systems and networks**; • **Software and its engineering → Petri nets**; *Abstraction, modeling and modularity*.

KEYWORDS

Distributed automation, Performance and reliability, Petri nets

ACM Reference Format:
Vuk Lesi, Zivana Jakovljevic, and Miroslav Pajic. 2019. Reliable Industrial IoT-Based Distributed Automation. In *IoTDI '19: Internet of Things Design and Implementation, April 15–18, 2019, Montreal, QC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3302505.3310072

1 INTRODUCTION

Recent advances in information technologies and embedded systems networking are revolutionizing manufacturing industry, leading to a new stage known as the Industry 4.0 [15]. Namely, to address fluctuating market demands, manufacturers have embraced mass customization (as opposed to mass batch production) and the personalization production paradigm [8]. This is achieved by ensuring high level flexibility and adaptability of manufacturing systems through rapid and cost effective changes in their structure, functionality and capacity [16]. In other words, there is a move toward Reconfigurable Manufacturing Systems (RMS) that can be ad-hoc re-configured to support a wide variety of products. The most prominent enabling technologies for RMS development are the (Industrial) Internet of Things (IIoT) and Cyber-Physical Systems (CPS) [13].

To ensure scalability, convertibility and customization, RMS should be modularly built on *IIoT-enabled smart things* - manufacturing field devices (sensors, tools, machines, ...); these represent CPS that in addition to the physical devices, integrate computation and communication to support a higher level of automation/autonomy. RMS capabilities, including modularity and re-configurability, impose new requirements on the control system design. The traditional automation pyramid (where each layer of devices strictly has a lower level of automation than the layer above) is broken up, giving way to control systems with functionality *distributed over different field devices that communicate with each other*. The functionality of an Industry 4.0 enterprise is based on ubiquitous communication between things (assets) that form the IIoT, and enable vertical, horizontal and end-to-end integration of manufacturing processes [1, 10].

Yet, reliable functioning of RMS with distributed control/automation tasks requires high performance connectivity of smart devices. Key performance indicators of the IIoT connectivity represent: (1) network availability (robustness to failures), (2) data loss and transmission errors, (3) data latency and jitter, and (4) data throughput [9]. For the functions of distributed industrial control systems, reliable communication, data latency and jitter are critical.

Design of industrial automation/control systems commonly comes in a form of a relatively low level of abstraction – i.e., as sequential discrete-event systems. In industrial practice, GRAFCET standard (IEC 60848) is frequently employed for functional specification of event-driven sequential control tasks. Control interpreted Petri nets (CIPN) are the formalism underlying GRAFCET, with the behavioral equivalence between these shown in [5]. Following this rationale, and considering the benefits that CIPN and the parent formalism of Petri nets (PN) provide for industrial automation, we have recently proposed a method for distribution of control tasks in industrial automation based on the use of CIPNs [12]. Specifically, starting from a CIPN representation of the global (i.e., centralized) control system, which may be extracted from a GRAFCET-compliant design tool, control system functionalities are automatically distributed to a number of local controllers (LC) executing on IIoT-enabled smart devices; local CIPNs, as well as control code in C, for all LCs are automatically obtained during this procedure. To ensure that the obtained functionality of the distributed system matches the one of the centralized system, coordination of the LCs with physical access to sensors and actuators is performed by means of communication.

IoTDI '19, April 15–18, 2019, Montreal, QC, Canada

ACM ISBN 978-1-4503-6283-2/19/04…$15.00
https://doi.org/10.1145/3302505.3310072

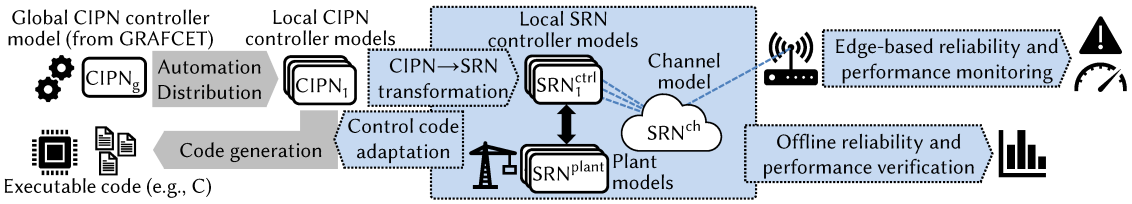


Figure 1: Methodology for reliable IIoT-based distributed automation – global (i.e., centralized) CIPN-based controller model is transformed into local CIPNs, each of which is used to generate executable code (in C) for smart device controllers. To enable system analysis for realistic environment (e.g., communication) models, the local CIPNs are further converted into an SRN-compatible representation; the channel and plant SRN models are introduced. Such models are used to verify system reliability and performance both at design- and run-time; any detected execution scenario that violates system requirements is then used to adapt the design of local CIPNs to avoid detected failure modes. The paper's contributions are highlighted in blue.

Yet, communication between smart devices raises the issue of reliable information exchange which is crucial in safety- and mission-critical production systems. GRAFCET and CIPNs are originally intended for modeling of centralized automation systems, and thus do not support inclusion of communication channel models for control design and performance analysis. Hence, control performance of the newly obtained distributed system, operating in realistic industrial environments, cannot be guaranteed. On the other hand, the parent formalism of PNs has been used for modeling and analysis of communication protocols [6], but stochastic properties of universally adopted wireless channels cannot be encoded within it.

Consequently, we bridge the gap between the expressiveness of the widely adopted control models and the need for verification of automatically synthesized distributed control systems. We achieve this by automatic translation of the distributed CIPN models into Stochastic Reward Nets (SRNs) — a variant of PN that supports stochastic timing features suitable for modeling of communication channels and time-varying physical executions; this enables capturing of automation-level (control-related) effects of realistic IIoT designs. SRNs were successfully used to model performance/reliability of software/hardware (e.g., [4, 18]), communication protocols (e.g., [14]), and distributed systems (e.g., [21]). We exploit composition of SRN models to verify relevant system properties, based on probability distributions obtained (and updated at runtime) from system measurements. Execution scenarios that violate the desired properties are then used to modify the distributed CIPN controllers such that potential failure modes are avoided. Finally, we do not limit the analysis to design-time only. To support dynamic IIoT environments, we develop an edge-based runtime monitoring system that checks properties of interest against system models any time the models are updated; SRN models are continuously updated using real-time process and communication channel measurements obtained by the smart devices and the monitoring system itself.

This paper is organized as follows. Sec. 2 provides an overview of state-of-the-art distributed automation modeling and design based on CIPNs as well as its limitations in realistic IIoT-based systems. Sec. 3 introduces SRNs, the transformation from the CIPN controller models, and network modeling. In Sec. 4, we present the use of such models for offline analysis and property verification, and introduce edge-based runtime verification of distributed automation systems based on SRNs. Sec. 5 presents the use of our framework on real-world industrial case studies, before concluding remarks in Sec. 6.

2 STATE-OF-THE-ART AUTOMATION DISTRIBUTION AND ITS LIMITATIONS

Effective implementation of RMS requires fast and reliable methods for distribution of control tasks to smart IIoT-enabled devices acting as local controllers (LCs). However, the system's control functionalities are commonly specified as centralized controllers using tools that produce global (i.e., centralized) controller description as a CIPN (e.g., GRAFCET-compatible tools). Hence, in this paper, we will employ a recently derived method for distribution of CIPN-based control tasks to LCs [12], which we outline in this section. We start with a brief overview of PNs and CIPNs.

Formally, a Petri net is a 5-tuple $\mathbf{PN} = (P, T, F, W, M_0)$, where $P = \{P_1, ..., P_m\}$ is a set of places (graphically presented by circles), $T = \{T_1, ..., T_n\}$ is a set of transitions (graphically presented by bars) such that $P \cup T \neq \emptyset$ and $P \cap T = \emptyset$, $F \subseteq \{P \times T\} \cup \{T \times P\}$ is the set of arcs connecting places and transitions, W is the vector of arc weights, and M_0 is the initial marking, denoting which places contain (how many) tokens at net initialization time [20]. At any time, the state of **PN** is defined by its marking, i.e., distribution of tokens (graphically presented by black circles inside places), while firing of transitions that removes tokens from some places and deposits them into others represents change of the **PN** state.

Control Interpreted Petri Nets (CIPN) are a variant of PNs where transition firing is synchronized to system inputs (i.e., sensor sampling) while control outputs (i.e., actuation commands) are issued from active places (i.e., that have a token). Formally, CIPN represents a 6-tuple $\mathbf{CIPN} = (P, T, F, C, A, M_0)$ in which P, T, F, and M_0 are the set of places, transitions, arcs, and initial marking, respectively as with PNs [5]. To synchronize sensor measurements with the CIPN, a set of logical conditions $C = \{C_1, ..., C_n\}$ is introduced; C_i represents a Boolean function of sensor values and it is allocated to corresponding transition T_i. Also, **CIPN** is synchronized with actuator outputs via a set of actions $A = \{A_1, ..., A_m\}$; A_i represents a set of Boolean (or some other) functions of actuator outputs allocated to places P_i. Note that all arcs in CIPN have weight 1, and in the initial marking M_0 only one place contains a token. In the graphical representation, conditions C_i (actions A_j) are denoted next to the corresponding transitions T_i (places P_j) (see e.g., Fig. 2). A transition T_i can fire and pass tokens to the succeeding places, if all preceding places contain a token, and condition C_i is fulfilled.

The CIPN-based method for distribution of control tasks to LCs, starts from the global **CIPN** that captures functional representation

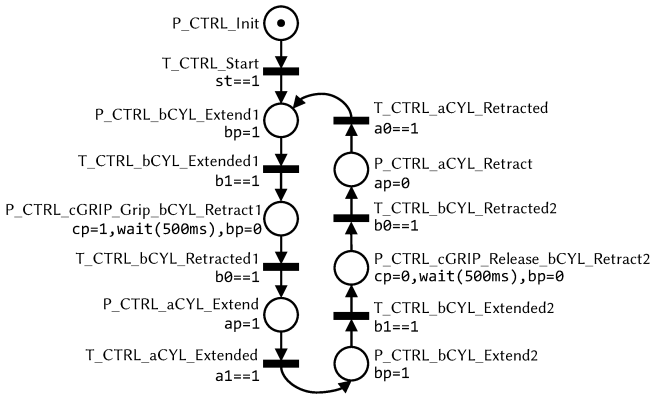


Figure 2: CIPN-based centralized control model of the pneumatic industrial manipulator shown in Fig. 3.

of the centralized control system, as well as the mapping of sensors and actuators to IIoT-enabled smart devices (i.e., their LCs). Then, the method automatically generates local **CIPN$_i$**, $i = 1, \ldots N$, for each LC using special-purpose graph-search algorithms. In addition to relevant logical conditions and actions assigned to places and transitions, each **CIPN$_i$** may contain communication commands (i.e., communication API calls) between LCs. These enable coordination and information exchange between LCs, to ensure that control functionality of distributed control matches the original **CIPN**. From **CIPN$_i$** functional description, C code for LCs is automatically generated. We illustrate this methodology on an example of pneumatic manipulator with two degrees of freedom (DOF).

Example. *A 2-DOF industrial manipulator in a pick-and-place configuration (Fig. 3) is used as a running example in this paper. The manipulator consists of two pneumatic double acting cylinders providing translational degrees of freedom, denoted A and B, and one pneumatic gripper denoted C. All actuators are controlled by electrically activated monostable dual control valves 5/2 (with 5 ports and 2 positions), triggered by signals* xp, x $\in \{a, b, c\}$. *Here, lower letter* x *denotes signals associated to the actuators denoted with capital letter X, $X \in \{A, B, C\}$. Also, cylinders are equipped with proximity switches that detect their final retracted (home) and extended (end) positions; home position sensor signals are denoted as* x0*, and end positions as* x1*,* x $\in \{a, b\}$. *The system is also equipped with a start switch* st*. Each actuator represents a smart field device with integrated LC with physical access to sensors and dual control valves. The allocation of LCs and signals is as follows: i) Cylinder A: $LC_1 \leftarrow \{ap, a0, a1\}$, ii) Cylinder B: $LC_2 \leftarrow \{bp, b0, b1, st\}$, and iii) Gripper C: $LC_3 \leftarrow \{cp\}$.*

*The manipulator operates as follows. On activation of the start switch (*st==1*), cylinder B extends (due to action* bp=1*). When cylinder B reaches end position (*b1==1*), the gripper is activated and grips the part at the picking position (*cp=1*). Then, cylinder B retracts (*bp=0*) and when it arrives to the home position (*b0==1*), cylinder A advances (*ap=1*). When cylinder A comes to the end position (*a1==1*), cylinder B starts extending (*bp=1*), and when it reaches the end position (*b1==1*), gripper C releases the part (*cp=0*) at the placing position. Then, cylinder B retracts (*bp=0, b0==1*), followed by retraction of cylinder A (*ap=0, a0==1*). The cycle is then automatically repeated without pressing the start switch. A timer ensures gripping and releasing of a part before retraction of cylinder B – i.e., a delay is inserted between commands* cp=1 *and* bp=0 *to secure gripping before raising the object from the picking position; similarly holds for placing.*

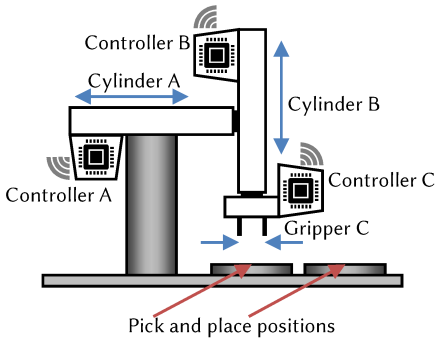


Figure 3: Running example — 2-DOF pneumatic manipulator in a pick-and-place configuration. Each of the cylinders and the gripper are controlled by a networked controller.

Fig. 2 shows a **CIPN** that captures the described functioning of the manipulator's global sequential behavior using the CIPN formalism. The use of the CIPN-based distribution of control tasks to LCs leads to local **CIPN$_i$** from Fig. 4. Each **CIPN$_i$** contains: (i) places and transitions with conditions and actions extracted from the global **CIPN** based on sensor/actuator mapping to LC_i, and (ii) arcs that ensure the desired control sequence through token passing. Furthermore, some of the places in **CIPN$_i$** contain commands for transmission (i.e., communication API calls) of certain signals from LC_i to LC_j. For example, place PaCTRL_TxB_aEnd contains command Send(B,a1), which sends information from LC_1 (cylinder A) to LC_2 (cylinder B) that sensor a1 is activated (i.e., a1==1 as cylinder A reached the end position); LC_2 receives this information at the transition TbCTRL_wfAextended.[1] Communication between LCs ensures that the distributed system behaves as globally specified.

However, unpredictable network delays and failures, including packet drops, may violate functional safety of the distributed system. For instance, if a message from controller B to controller C fails to reach controller C (in time), the part ends up ungripped and the workcycle is wasted, or collision occurs between the gripper and the part. Also, if a cylinder fails to inform the other that it has reached the end position, the entire system is deadlocked. Note that such scenarios cannot occur with centralized control.

Consequently, to ensure reliable system operation in realistic IIoT environments, it is necessary to analyze distributed control performance when realistic network and faults models are taken into account. To achieve this, we first propose the use of Stochastic Reward Nets (SRN) to model communication and time-variable executions; the formalism supports modeling of probabilistic properties of communication channels/execution delays in a framework suitable for quantitative reliability and control-related performance analysis. We show how physical components (i.e., the controlled plants) and communication channels can be modeled with SRNs, and how the distributed CIPN-based controller models can be automatically transformed into SRN models. We also show how the

[1] We use the abbreviated descriptive notation for places and transitions to improve model readability; e.g., transition TbCTRL_wfAextended on controller B waits for cylinder A to extend, while place PaCTRL_TxB_aEnd on controller A implements sending of signal a1==1 to controller B.

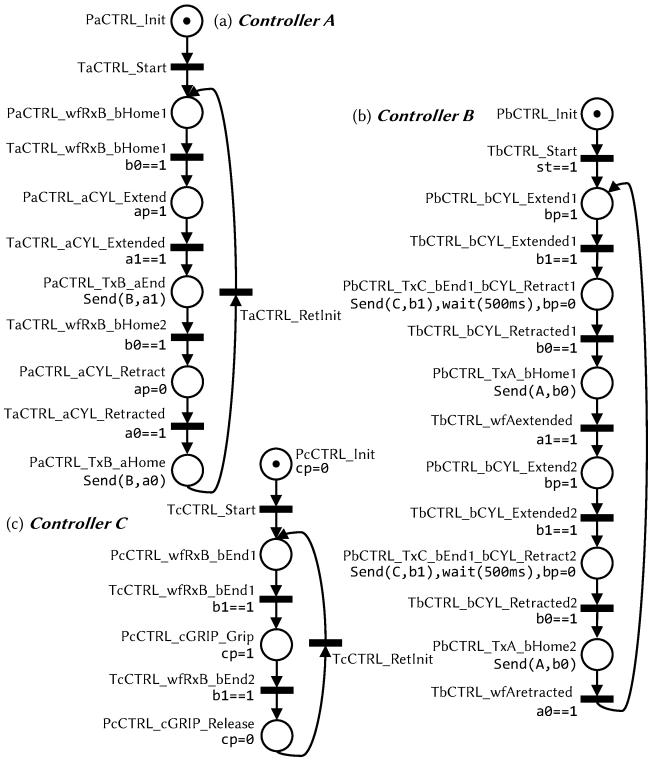


Figure 4: CIPN-based distributed control model of an industrial manipulator from Fig. 3 (obtained from CIPN in Fig. 2).

desired system properties can be specified and efficiently verified with tools that solve SRN, and how to avoid detected failure-modes by adapting local $\mathbf{CIPN}_i$ controllers. As deployment environments of IIoT-based RMS are dynamic, we also introduce an edge-based reliability and performance runtime monitoring system that re-evaluates properties at runtime based on current system models. In what follows, we describe the stages of the framework from Fig. 1.

3 NETWORKING-AWARE MODELING FOR DISTRIBUTED SEQUENTIAL CONTROL

Stochastic Reward Nets (SRN) extend the expressiveness of PNs by introducing stochastic transition firing times, and *race policies* for resolving firing of competing transitions. Additionally, general marking-dependent guard functions and firing rates can be specified. Moreover, SRN support marking-dependent reward functions that encode an arbitrary measure of system performance, enabling extensive reliability and performance analysis [22].[2]

To analyze effects of non-zero execution, network transmission delays, and packet drops on distributed sequential control, we transform the distributed control specification given by $\mathbf{CIPN}_i$, into SRN models $\mathbf{SRN}_i^{ctrl}$, $i = 1, ..., N$; we also add corresponding $\mathbf{SRN}_i^{plant}$ models of the physical components controlled by $\mathbf{SRN}_i^{ctrl}$, and $\mathbf{SRN}_j^{channel}$, $j = 1, ..., M$, modeling used communication channels.[3]

[2]The PN definition from Sec. 2 can be formally extended to capture features of SRNs; for more details and complete theoretical review, refer to Ch.12 of [22].

[3]This supports the general case when more than one communication channel is used for information exchange between controllers; e.g., frequency separation may be used in such wireless networks to minimize communication contention between the devices.

Due to the structure of CIPN and SRN formalisms, the translation of the local controller models $\mathbf{CIPN}_i$ to $\mathbf{SRN}_i^{ctrl}$ is straightforward for all places and transitions, except:

(1) Places issuing actuation commands by setting a control variable, e.g., `actr=value`, and transitions conditioned by the state of an input signal (sensor value), e.g., `sensor==value`,
(2) Communication API calls for signal transmission in places, i.e., `Send(destination,signal)`, and for waiting for a signal on transitions, i.e., transition conditions `signal==value`.

In essence, exception (1) refers to explicit modeling of the interaction between the controllers $\mathbf{SRN}_i^{ctrl}$ and the plant $\mathbf{SRN}_i^{plant}$. The implicit, operating system-supported, sampling of sensors and issuing of actuation commands in the paradigm of CIPNs must be transformed into the SRN setting such that analytical or simulative solutions of the composition of controller and plant SRNs is supported. In addition, exception (2) pertains to modeling of the coordination between controllers that is mediated over the channel $\mathbf{SRN}_j^{channel}$. In this case, the implicit transmit/receive paradigm of CIPNs must be transformed into explicit synchronization of controllers via a channel model that are to be composed and solved.

To address these challenges, in the rest of the section we present methods to model the physical components, network, and controller-network interaction, while Sec. 4 focuses on solving the SRN model.

3.1 Modeling Physical Components

We start by describing an example model of physical plants being controlled by the distributed automation system, as well as the interaction between the plants and distributed controllers, with emphasis on SRN-enabled modeling conveniences.

3.1.1 Plant Modeling. Expected plant behavior is usually available as properties of the controlled process (i.e., plant) are known at design-time. On our running example, we show how an SRN-based model of the plant can be developed. Fig. 5(a) shows the **SRN** model of the two-position linear pneumatic cylinder from Fig. 3. As the position of the cylinder cannot be controlled anywhere between the two end positions, places `PaCYL_Retracted` and `PaCYL_Extended` are sufficient to model the two end positions (states) (see Fig. 5(a)).

The cylinder is initially in the retracted (home) position; hence one token is initially placed in `PaCYL_Retracted`. The transition `TaCYL_Extending` (`TaCYL_Retracting`) is a timed transition that models the delay between the extend (retract) actuation signal and the cylinder reaching the end (home) position. If such delay is fixed (i.e., does not change during system operation), a deterministic firing time transition can be utilized, as in Fig. 5(a).[4] On the other hand, the pneumatic cylinder may not exhibit fully deterministic behavior in terms of travel time between the end positions. In such cases, the deterministic time transitions can be replaced with a distribution that models travel time variations around the nominal travel time.[5]

In general, the plant side of the controller-plant interaction is modeled by guarding transitions in the plant model with functions dependent on the marking of the controller model. In our example,

[4]Note that timed transitions are denoted as rectangles, while immediate transitions are denoted using the usual bars as in regular PNs.

[5]For example, a common method is to use Erlang-k distribution, where each of the k exponential stages have average firing time $\frac{nominal\ travel\ time}{k}$. Another alternative is to use the uniform distribution, as in the real-world case studies described in Sec. 5.

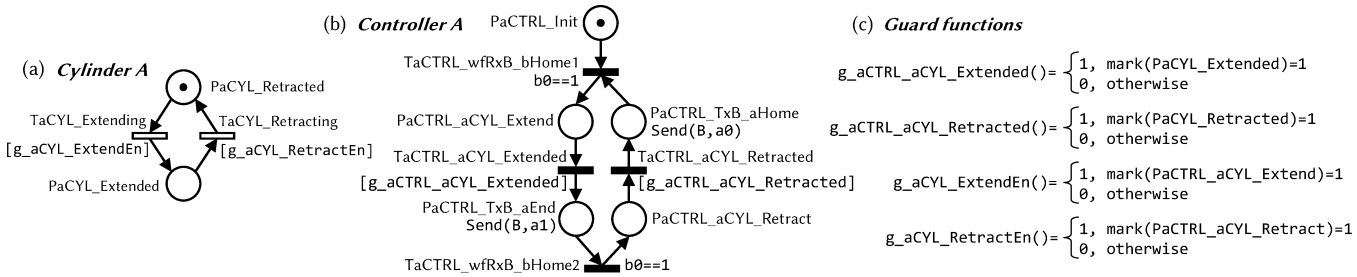


Figure 5: Modeling the plant and plant-controller interaction: (a) SRN model of a two-position pneumatic cylinder; (b) extended model of the controller for cylinder A from Fig. 4(a) (the model is not SRN as it still maintains communication API) – plant-controller interaction is captured through transitions guarded by Boolean functions (c) of the plant and controller markings.

transition `TaCYL_Extending` (`TaCYL_Retracting`) is guarded by a Boolean function `g_aCYL_ExtendEn` (`g_aCYL_RetractEn`) – i.e., the transition is enabled if there is a token in the corresponding place in the controller model where cylinder extending (retracting) is commanded (Fig. 5(c)). The following subsection introduces the controller side of the controller-plant interaction.

Remark 1. *Since CIPNs are the de-facto standard for representation of industrial automation controllers, in this work we assume that a PN-based model (e.g., a CIPN or SRN model) of the physical plant is available. On the other hand, if plant models are given as automata, timed-automata or discrete-event systems, which are other formalisms widely used to model dynamics of physical systems in industrial automation, standard methods for translation of such formalisms into PN-compliant models (e.g., as in [3, 7]) can be used.*

3.1.2 Modeling Controller-Plant Interaction. In general, places issuing actuation commands in the form `actuator=value` in an initial CIPN controller model, are transformed into places in the corresponding SRN controller model whose number of tokens is assessed by the aforementioned guard functions in the plant model. For example, to map command `ap=1` in place `Pa_CTRL_aCYL_Extend` from the CIPN in Fig. 4(a), the cylinder's transition `TaCYL_Extending`, in Fig. 5(a), is guarded by function `g_aCYL_ExtendEn` that returns 1 if the cylinder's controller (Fig. 5(b)) is in place `PaCTRL_aCYL_Extend` (i.e., where actuation signal to extend is commanded), and otherwise returns 0. This effectively models the actuation part of the controller-plant interaction. Formally, the controller deposits tokens in places that enable plant transitions; semantically, the controller sends actuation commands to locally connected actuators.

Similarly to the described actuation modeling, in the controller model we introduce guard functions on the plant model's state (i.e., token distribution) to model plant sensing. Formally, each transition in the CIPN controller model conditioned by `sensor==value` is replaced with an immediate transition guarded by a Boolean guard function that evaluates to 1 if the plant model currently has a token in the place corresponding to the state where the given sensor's reading is equal to the specific value, and otherwise to 0. Note that more than one plant's place may be considered simultaneously for complex transitions (i.e., that use multiple sensors). Extensions to real-valued variables and more complex guard functions are also supported, but are not considered here to simplify our presentation.

To illustrate this, the intermediary controller model corresponding to the CIPN model of the controller for this cylinder (Fig. 4(a)) is shown in Fig. 5(b), while Fig. 5(c) contains the corresponding guard functions. Transition `TaCTRL_aCYL_Retracted` guarded by `a0==1` in Fig. 4(a) is mapped into transition `TaCTRL_aCYL_Retracted` in Fig. 5(b) guarded by function `g_aCTRL_aCYL_Retracted`. Note that this controller model includes only the interaction between the controller and the plant, and carries over the CIPN communication paradigm — transmissions `Send(destination,signal)`, and signal receptions modeled by transitions guarded by `signal==value`.

3.2 Modeling Network-Controller Interaction

In the CIPN models of distributed controllers, communication between a transmitter and a receiver is handled via a place issuing a `Send(destination,signal)` command on the transmitter, while the receiver guards the transition from its place preceding reception with a corresponding `signal==value` condition, as illustrated in Fig. 6(a). A suitable SRN-compatible representation is needed for this place/transition to enable interaction with the communication channel. Yet, explicit network modeling is (a) medium-, (b) protocol-, and (c) implementation-dependent. For instance, Fig. 6(c) shows the model of a half-duplex, acknowledge-required unicast CSMA-CA-based communication channel. While other channel models can be as easily adopted, in this paper we define the transformation of the sending/receiving semantics from CIPNs into an SRN-compatible representation for this specific channel model. Note that while seemingly simple, this model captures application-level (control-related) effects for the experimentally observed communication delays/faults in IIoT deployments. Additionally, as we will show in Sec. 5, this abstraction was successfully used to discover flaws in existing implementations of distributed automation systems.

The considered channel can either be idle, busy transmitting a communication packet, just completed the transmission in preparation to take the acknowledgement (ACK), or busy transmitting an ACK, as shown in Fig. 6(c). The transition `Tch_wfTx` is enabled only when the respective transmitter is in a corresponding transmit state. Thus, guard function `g_Tx` is dependent on the number of tokens in `Pa_wfTx` place on the transmitter (i.e., where the transmitter is issuing packet transmission, as shown in Fig. 6(b)).

In the general case with multiple transmitters, the transmitter that first deposits a token into its transmit place (i.e., initiates transmission), triggers a change in the channel state. Race conditions between transmitters can be resolved deterministically (e.g., by priorities), or probabilistically. Once `Tch_wfTx` transition is enabled, the token in the channel model can transition to place `Pch_TxBusy`

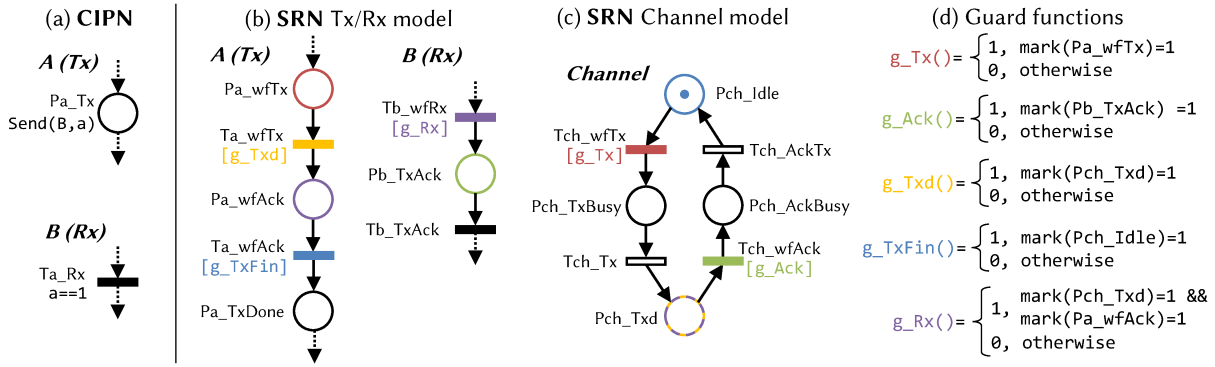


Figure 6: SRN model of a half-duplex, ACK-required unicast CSMA-CA channel. Places model channel states, immediate transitions are enabled when packet (ACK) transmission is enabled on Tx (Rx); stochastic firing time transitions model the delay.

(i.e., the channel changes state), where it will dwell until the stochastic time transition Tch_Tx modeling the network/channel-induced delay fires. Also, once Tch_Tx fires, the packet is sent, enabling the transition Ta_wfTx, advancing the transmitter into wait-for-ACK state. Hence, as the channel is ready to transmit the ACK (token in Pch_Txd), and the transmitter is waiting for the ACK (token in Pa_wfAck), the receiver receives the packet, advancing to Pb_TxAck place (i.e., Tb_wfRx fires). This captures packet transmission model.

After a message transmission, the channel model advances to Pch_AckBusy state, as the transition Tch_wfAck is enabled. Immediately, the receiver continues execution as the ACK is handled by lower layers of the communication stack. Consequently, the receiver's token advances through Tb_TxAck, which concludes the receiver's activity in the packet exchange. On the other hand, the transmitter is blocked at Ta_wfAck by the guard g_TxFin, until the ACK propagates through the channel. The channel is busy transmitting the ACK until the stochastic firing time transition Tch_AckTx fires, returning the channel into idle state, and advancing the transmitter into Pa_TxDone, which concludes acknowledging.

Note that in addition to modeling of event sequencing, the SRN models allow for capturing time-to-transmit and time-to-ACK, which we model using two independent stochastic-time transitions. This enables adoption of realistic models where an ACK packet propagates much faster, as it is much shorter than the original packet, and is immediately preceded by a successful full-packet transmission. Complete transmitter/receiver and channel models are shown in Fig. 6(b)-(d). For improved readability, transitions and places are colored according to their dependencies in Fig. 6(b) and (c).

Furthermore, this model imposes two race conditions, between transitions Tch_wfTx and Ta_wfAck, and between Tch_wfAck and Tb_TxAck. In both scenarios, the competing transitions are simultaneously enabled, and firing of the former disables the latter (simultaneous transition firing is not allowed in PNs). In the first case, when transmitter A receives the ACK, guard g_TxFin no longer inhibits Ta_wfAck, as the channel is in Pch_Idle state. However, the token in Pch_Idle also enables another transmitter A' to change the state of the channel from Pch_Idle to Pch_TxBusy, which in turn disables Ta_wfAck. In the second case, as the receiver is ready to advance through ACK transmission (Tb_TxAck is enabled), and the channel to start transmitting the ACK (Tch_wfAck is enabled), firing of Tb_TxAck disables Tch_wfAck as it removes the token from Pb_TxAck; this causes the guard g_Ack to inhibit Tch_wfAck.

We address with transition priority assignment; if prio(Ta_wfAck) >prio(Tch_wfTx) and prio(Tch_wfAck)>prio(Tb_TxAck) the race conditions are resolved properly; this is semantically correct as the first transmitter must complete transmission before another one can initiate the next transmission in CSMA-CA-based networks (first scenario), and the receiver has to initiate ACK transmission upon packet reception before continuing execution (second scenario).

Remark 2. *The presented modeling approach is flexible, and can be adjusted for different PHY- and MAC-layer variations. For instance, if an ACK is not required for low-criticality transmissions, the above presented channel model can easily be modified by eliminating places and transitions modeling ACK transmission/reception. Low-level protocol features, such as multiple-retry schemes, can also be accounted for. For instance, standard exponential-based retransmission back-off can be modeled by probabilistically flushing tokens from* Pch_TxBusy *place in case of transmission failure, and depositing new tokens at a stochastic rate. Other random back-off models are also allowed.*

Furthermore, in the presented model, the receiver advances to transmitting the ACK through Tb_wfRx *transition that is guarded by* g_Rx *dependent both on the channel and the transmitter markings. This ensures that, in the case of multiple receivers, only the one targeted by the active transmitter advances execution; i.e., the channel is unicast. The guard function* g_Rx *can easily be modified to model multicast or broadcast channels. For presentation clarity, we consider the communication model from Fig. 6 throughout the paper as it applies to the network underlying our case study. For details on the use of PNs for modeling of communication and cooperation protocols refer to [6].*

Example (Industrial Manipulator with 2-DOF). *We used the presented methodology for translation of distributed CIPN controllers into SRN models that also capture communication (i.e., networking) between the controllers, to fully transform the CIPN model from Fig. 4 into an SRN model shown in Fig. 7. The following section analyzes the model utility by introducing formally-encoded functional and safety properties that can be verified in tools that support analysis of SRNs.*

4 SYSTEM ANALYSIS AND VERIFICATION

The developed SRN-based models can be used to verify desired system properties. Given the probability distributions of network

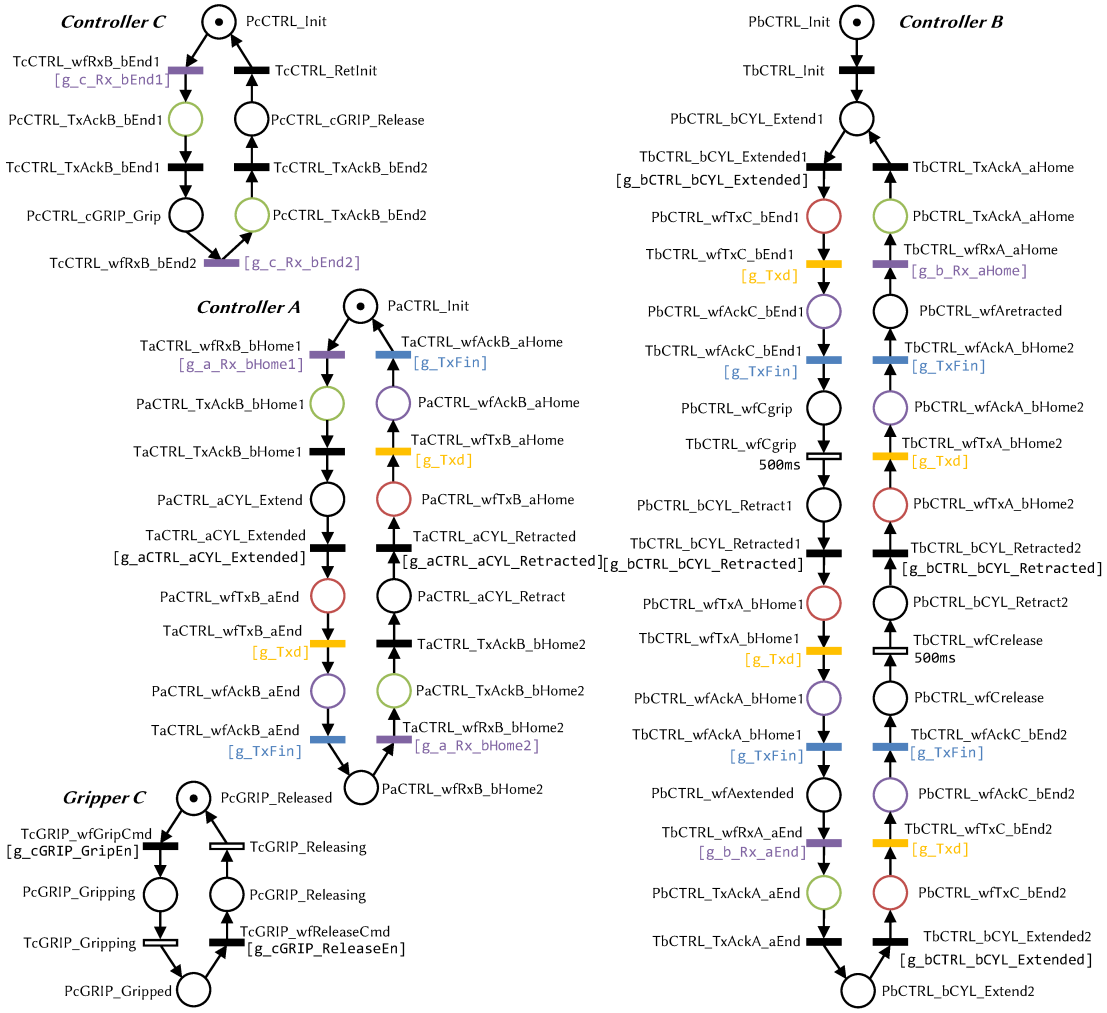


Figure 7: SRN-based distributed control model of a simple industrial manipulator; models of cylinders A and B, as well as the channel model are shown in Fig. 5(a), and Fig. 6(c), respectively; guard functions are defined as per the pattern shown therein.

delays, actuator and plant response times, as well as controller settings as inputs, PN software packages can be used for system analysis. If all timed transitions are either immediate or have exponential firing time, the model can be solved using analytic/numeric methods [11, 22]. On the other hand, for generally distributed firing times, discrete event simulation is performed. *As assuming exponentially distributed or immediate firing times is not valid in most IIoT applications*, we use the Stochastic Petri Net Package (SPNP) [11] that also supports discrete-event simulation for quantitative system analysis. We now discuss specification of relevant system properties as well as employed design- and run-time verification setup.

4.1 Specifying System Properties

Safety Properties. Safety properties can be specified as reward functions — arbitrary functions of the controller and plant states (i.e., markings) that are evaluated at every marking update (i.e., discrete simulation step). For instance, for our running example, gripper C must never be closed while cylinder B is advancing downward towards the part; this leads to collision between the gripper and part. If `mark(PcGRIP_Gripped)==1 AND mark(PbCTRL_bCYL_Extend1)==1` holds, the safety property is violated. Thus, a Boolean reward function that returns 1 when the condition holds, and 0 otherwise is used to evaluate probability of occurrence of such an event; any nonzero such probability is used as an indicator about potential safety violation. To simplify our notation, we denote functions of this form as `R()=if(BooleanCondition,ValueIfTrue,ValueIfFalse)`.

For our running example, SPNP returns that the system is safe when the network-induced communication delays are exponentially distributed (rate parameter used for packet delays is 1/200 while it is 1/1000 for ACK). In Sec. 5, we use actual channel measurements acquired from a deployed system for analysis using the SPNP tool.

Liveness Properties. Besides safety, it is important to ensure the system makes forward progress and performs according to desired performance specification; we somewhat abuse the notation and refer to these as liveness properties. Analysis of SRN models results in a set of average measures for each place and transition. For places, average number of tokens and average probability of a place being

non-empty can be defined. These measures provide insight in the system dwell time in a particular state. For example, we expect that the channel dwells zero time in Pch_Txd place as otherwise would indicate a problem with ACK transmissions (this can be verified by analyzing the model within SPNP). Also, actuation systems are commonly provisioned by specific upper bounds on the actuator active times. In our running example, probability that the pneumatic cylinders are either in the state of extending or retracting can be used to compute average duty cycles of the mechanical components, which can be used to verify that the system is operating within prespecified limitations, or anticipate increased wear.

Conversely, average throughput of tokens through each of the transitions can be defined, as well as probability that a transition is enabled. Transition throughput can then be used as a performance metric; for example, the speed at which the pick and place cycle is performed for our running example – analysis using SPNP reveals that the throughput of all controller transitions is $0.295\ \frac{tokens}{sec}$ resulting in the processing time of $3.389\ \frac{sec}{unit}$.[6]

4.2 SRN-based System Analysis

The derived SRN models are used for design- and run-time system's safety and performance analysis using existing PN-supported tools.

Capabilities of Petri Net-Based Tools. When analytic/numeric methods are utilized, violation of safety properties can be determined at the time of construction of the reachability graph; i.e., the undesirable marking(s) can be identified without statistical simulations. On the other hand, if discrete-event simulation is performed (e.g., in the case of non-exponentially distributed firing times), reachability of markings where a safety property is violated is determined on per-simulation-run basis; i.e., obtained results translate into probabilistic guarantees. Two broadly used PN-based tools are [2, 11], which implement a type of stopping criteria both for analytic/numeric and simulative solution modes. Therefore, if a safety property violation is detected (i.e., non-zero probability of reaching undesired state), the analysis can be halted.

Runtime Monitoring Support. Various runtime effects such as communication channel utilization, component aging, and mechanical wear may affect the parameters of the developed model; thus, results of design-time (i.e., offline) system analysis based on initially obtained SRN models may not continue to hold during system execution. Hence, the ability to check for statical property violation at runtime is crucial for reliability and performance monitoring. We explored runtime support for monitoring of distributed automation systems; the stopping criteria feature of PN-based tools is very useful for runtime safety verification as the supervisor can be warned on predicted unreliability, before the full analysis is completed.

We developed an execution environment in which process measurements (e.g., cylinder travel times) and channel features (e.g., packet propagation delays) are acquired in real-time, and used for runtime adaption of the prespecified parameters of probabilistic features within the SRN model. To ensure timely response in bandwidth-constrained IIoT-based deployments, we explore execution of the model checking tool on the edge, rather than the cloud (details are provided in Sec. 5). Edge-based monitor deployment is naturally promoted for IIoT-based automation, as continuously powered gateway devices that support higher-level coordination and decision-making exist by design in such systems.

[6]Given the specific structure of PNs modeling controller behavior, the throughput of all transitions is the same as no tokens are being generated or flushed at runtime.

Remark 3 (Estimating distributions vs. directly using sample-based measurements). *State-of-the-art PN tools support parametric specification of transition firing times conforming to most well-studied probability density functions (more than 15 distributions), suitable for flexible modeling of discrete event processes [2, 11]. In this setting, distributions are fitted to offline data to create an accurate model of the real system. Still, state-of-the-art tools also support sample-based firing times — i.e., the solver can sample firing time arrays during simulation that are obtained from measurements at runtime, rather than generating a random sample from a predefined distribution [11]. Our system is capable of exploiting this feature such that the additional step of fitting a probability distribution to the acquired data can be eliminated, increasing the statistical fitness of the obtained measures to the real system. Consequently, while analysis of a model fitted with offline date is, our online edge-based performance and reliability monitoring is not dependent on the IID assumption as it employs process and channel measurements acquired at runtime.*

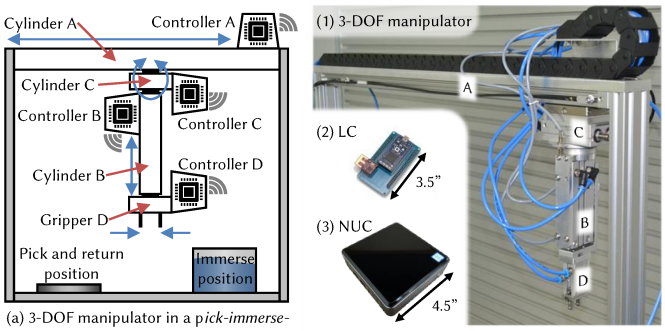
5 INDUSTRIAL CASE STUDIES

On two real-world industrial case-studies, we show applicability of our methodology for IIoT-enabled distributed automation: (I) 3-DOF pneumatic manipulator, and (II) a complex pneumatic manipulator with parallel processes. The considered manipulators are not classical; they are modularly designed in terms of mechanical subsystems and their control (using a smart IoT device), to facilitate reconfiguration. Also the considered control scenarios do not follow the conventional IEC 62264 hierarchical industrial automation pyramid. While we limited our evaluation to manipulators in our physical testbed, our approach applies to other IIoT-enabled equipment.

In both case-studies, we start from distributed control models obtained using existing techniques. We transform these CIPNs into SRNs and perform analysis with the developed plant and channel models. We first show how to discover potential problems that arise with the use of existing distributed controllers executing in realistic IIoT environments, before showing how these can be addressed through patches in the code generation stage (by introducing suitable communication constructs and exploiting LC runtime support).

5.1 Case Study I: 3-DOF Industrial Manipulator

5.1.1 Physical Setup and Modeling. The considered manipulator is configured to pick a part at the picking location, transfer it to the immersion location and immerse it into liquid, retract the part from liquid, shake of excess liquid (by means of rotation), and return the part to the original location. The pneumatic cylinder configuration with highlighted components is shown in Fig. 8(a), while the top portion of the system (i.e., cylinders) are shown in Fig. 8(b1). Cylinders A and B, providing translational DOF, as well as C, providing the rotational DOF, are all equipped with end-position sensors, while the gripper D is not (similarly to our running example). Each cylinder modules is controlled by a low-cost ARM Cortex-M3-based NXP LPC1768 microcontroller (i.e., LC in Fig. 8(b2)) clocked at 96 MHz. All LCs execute the C code generated from the formal CIPN description using our framework from [12]. Finally, LCs form a low-power, low-latency network via IEEE 802.15.4-compliant radio modules.



(a) 3-DOF manipulator in a *pick-immerse-shake-return* configuration

(b) System components

Figure 8: Case Study I: (a) 3-DOF industrial manipulator; (b1) Cylinders A and B provide translational DOFs, C provides rotational DOF, gripper D handles parts; each is controlled by a low-cost ARM Cortex-M3-based LC (b2). Size- and power-aware NUC (b3) monitors system's performance (pneumatic cylinders do not scale to the LC and the NUC).

Cylinder	Forward stroke bounds		Reverse stroke bounds	
	Lower [ms]	Upper [ms]	Lower [ms]	Upper [ms]
A	1100	1120	689	715
B	107	143	146	172
C	119	130	111	122
D	45	55	45	55

Table 1: Travel times for the cylinder are modeled using uniform distribution with parameters specified in the table.

Local CIPN controller models, obtained using the existing framework from [12], are transformed into SRN-compatible representation, and plant and channel models are developed as described in Sec. 3. Plant model parameters (i.e., cylinder travel times) are obtained from experimental measurements as they depend on a number of external parameters (e.g., control valve setting, nominal air pressure). Therefore, in the SRN models, cylinder response times (i.e., travel times from home to end positions) are modeled using the uniform distribution with parameters given in Table 1.[7]

Similarly, channel parameters (i.e., propagation delay distributions) were obtained from the data acquired from the physical setup. Fig. 9 shows the histograms of acquired transmit-to-receive and receive-to-ACK times in experiments executed over two days, illustrating small changes when the system operates in nominal conditions. Truncated normal distribution[8] is adopted with parameters $\mu = 4.84\ ms$, $\sigma = 4.7 \cdot 10^{-6}$, $x_{min} = 4.83\ ms$, $x_{max} = 4.85\ ms$ for $Tx \rightarrow Rx$ and $\mu = 0.57\ ms$, $\sigma = 4.7 \cdot 10^{-6}$, $x_{min} = 0.56\ ms$, $x_{max} = 0.58\ ms$ for $Rx \rightarrow Ack$ times; these parameters were used for offline system verification. To also cover the scenarios where environmental and operating conditions may significantly change, probability distribution parameters were updated at runtime based

[7]The case study model is substantially more complex than the running example model. However, the CIPN/SRN model schematics are omitted due to their close semantical similarity with the running example.

[8]Truncated normal distribution is a variant of the Gaussian normal distribution that allows the random variable to be bounded from bellow and/or above. In addition to the model being realistic as a communication packet will never have zero or infinite propagation time, our experiments showed that the distribution was able to fit the obtained experimental data very well.

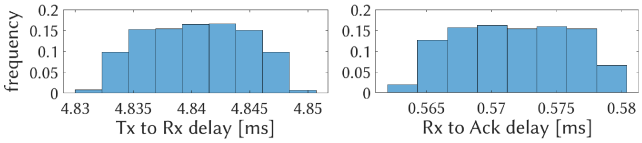


Figure 9: Histograms of packet and ACK propagation delays measured over two days from our industrial testbed.

on acquired delay measurements, and the properties reverified on an edge-based near-gateway monitor device, as described below.

5.1.2 Verification and Runtime Monitoring. We run SPNP on an Intel Core i3-based NUC platform, shown in Fig. 8(b3), which is suitable for edge-based near-gateway deployment. The NUC platform monitors the deployed industrial manipulator by collecting event-triggered communication packets between distributed controllers, and receiving additional diagnostic measurements from controllers themselves (e.g., cylinder travel times, actuation to sensing delays), obtained from local on-line monitoring using the Microchip ZENA IEEE 802.15.4-compliant adapter.

We considered an exhaustive list of safety and liveness properties prescribed by a domain expert. To illustrate our approach, in this paper we focus on the following properties:

Property 1 (Gripper D is always gripped during cylinder A stroke). *Violation of this property implies dropping of the part during transfer between the picking/placing position and the immersion position causing damage to the part and/or the manipulator platform. We add the reward function indicating violation of this property as:*

```
R1()=if((mark(PdGRIP_Gripped)==0 AND
         mark(PaCTRL_aCYL_Extend)==1) OR
        ((mark(PdGRIP_Gripped)==0 AND
         mark(PaCTRL_aCYL_Retract)==1)),1,0).
```

Property 2 (Gripper D is always released during cylinder B part-approaching stroke). *Violation of this property implies collision between the gripper and the part. The corresponding reward function indicating property violation is specified as:*

```
R2()=if((mark(PdGRIP_Released)==0 AND
         mark(PbCTRL_bCYL_Extend1)==1,1,0).
```

Property 3 (Cylinder C duty cycle is at least 5.5%). *Violation of this property implies abrupt rotation of the part after immersion causing increased wear of mechanical components. The probability of places* `PcCTRL_RotateCW` *and* `PcCTRL_RotateCCW` *collectively represent the fraction of time cylinder C is rotating.*

Property 4 (Manipulator processes parts at the maximum rate of 3 $\frac{sec.}{unit}$). *Violation of this property implies increased mechanical wear and potential issues with feeding surrounding manufacturing systems at a rate higher than expected. The average throughput of any of the controllers' transitions is indicative of this liveness property (i.e., the rate at which controllers visit their initial state equals the rate at which parts are handled).*

Furthermore, we consider the following representative scenarios, which were observed during system operation.

Scenario 1 (Nominal system operation). *This scenario pertains to offline verification of prescribed properties using nominal plant*

parameters, and the acquired channel model corresponding to experimental measurements at the time of deployment (i.e., parameters from Sec. 5.1.1); our experiments showed that the model parameters would not significantly change in nominal operating conditions.

Scenario 2 (Congestion-induced network unavailability). *Due to increased channel utilization, network packets may be delayed or dropped at higher than nominal rates; our experiments showed that bimodal (or multimodal) distributions may occur in such conditions.*

Scenario 2 was observed when a large number of additional LCs was introduced, all using the same wireless channel for communication. To realize this scenario and increase channel utilization in a realistic manner (in terms of channel temporal and bandwidth requirements), we emulated additional pneumatic manipulators similar to the real manipulators considered here – i.e., a mixture of the running example and case study I/II manipulators were implemented with real controllers, but software-emulated cylinders.

Scenario 3 (Component Fault – Gradually increasing air pressure due to a fault in the air pressure control system). *Due to a fault in the air pressure control system, increased air pressure is delivered to the pneumatic cylinders causing gradual shortening of cylinder response times (i.e., travel times between end positions).*

We implemented Scenario 3 by gradually increasing air pressure in the pneumatic system to emulate a system component fault.

5.1.3 Results. Table 2 shows results of model analysis[9] for these properties. Average model analysis runtime in SNPN, capturing 1000 discrete-event simulations of 1000 *sec* duration ($>$ 200 manipulator cycles), on the NUC is 28.2 *sec*. As the centralized control system is correct by design, and wireless communication does not incur significant delays and packet drops between local controllers under nominal conditions (Scenario 1), the system performs as expected (average cylinder C duty cycle is 6.1208%, average part processing time is 3.9368 *sec*.), and safety properties are not violated. An acquired sequence of events during a sample of the manipulator run under nominal conditions is shown in Fig. 11(top).[10]

In Scenario 2, as gripper D is not equipped with end-position sensing (due to its size constraints), communication between controllers B and D is most-vulnerable; by design, controller B sends a grip command to controller D and waits deterministically for 500 *ms* assuming that the operation has finished after this time elapses. If the communication packet fails, for instance, controller D may leave gripper D released when it is supposed to grip the part before cylinder B retracts it from the picking position, as shown in Fig. 11(bottom) during an experimental run in this scenario.

From our runtime analysis, we observed that protocol-level retries are insufficient to provide reliable packet delivery. Generally, the channel model presented in Fig. 6(c) does not cover unsuccessful transmissions, which acount 3% of all packet transmissions. By running the analysis on the derived model, we obtain significant probabilities of violation of properties P1 and P2 (26.9% and 3.3%, respectively). In the case of controllers A, B, and C, unsuccessful communication causes a deadlock; in essence, receiving controllers

[9]Our analysis is based on discrete-event simulation supported by SNPN, as our model includes non-exponentially-distributed transitions. Recall that if all transitions are either immediate or exponential, analytic/numeric solutions can be obtained.

[10]Timing diagrams were obtained by directly measuring sensing/actuation signals.

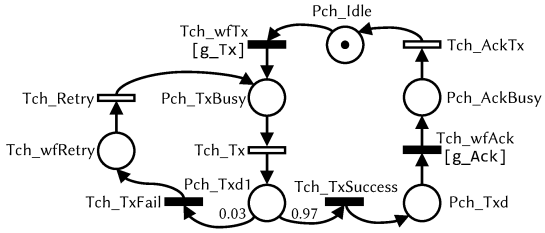


Figure 10: SRN model of a half-duplex, ACK-required unicast CSMA-CA-based communication channel with explicit delayed retries. In this specific case, 97% of transmissions are initially successful, while 3% are retransmitted at least once.

are eternally left waiting for a signal to continue execution, and the (unsuccessful) transmitter exhausts protocol-level retries. Cylinder C average duty cycle and the production rate are omitted in Table 2, as the system is often halted due to a communication interruption.

We thus employ the reasoning enabled by the presented SRN model analysis to modify the translation of the CIPN's communication semantics; we do this during code generation for LCs in order to include application-level transmission retries. Consequently, we also redesign the channel model to account for this change; Fig. 10 shows the improved channel model from Fig. 6(b). We redeployed the entire distributed system, using the modified semantics for communication APIs in LCs, and reproduced Scenario 2. We showed that in this case, despite network congestion, the system operated safely with marginal performance degradation – i.e., single part average processing time increases by ~ 0.03 *sec* due to application-level retransmission-handling delays. Note that the probability of successful transmission is updated at runtime (as shown in Sec. 4.2), to eliminate pessimistic results in non-congested deployment scenarios, and to maximize analysis accuracy under dynamic runtime conditions. In addition, while the discrete-event analysis by SPNP tool takes 28.6 *sec* on average on the NUC, asserting possible safety violation takes only 313.6 *ms* (an order of magnitude lower than the system's cycle time), which prevents faulty system operation.

Finally, in Scenario 3, due to the gradually increasing air pressure, parts were processed at a higher average rate, and cylinder C duty cycle was reduced, potentially endangering process quality and increasing component wear. Over time, part processing time is reduced to 3.2154 *sec* and cylinder C duty cycle decreased to 5.6214%, as shown in Table 2. Discrete-event model simulation takes an average of 29.7 *sec* on the NUC using incremental channel and process measurements, and warns the supervisor that the system's performance is degrading. This information is used to correct the fault before system performance degrades below acceptable limits.

5.2 Case Study II: Industrial Manipulator with Parallel Processes

The manipulator is configured in a parallel loading/unloading configuration; it consists of two (parallel) cylinders A and B, and a rotary cylinder C. The rotary cylinder rotates a base with parts, while cylinders A and B load/unload parts in parallel. The CIPN model of the centralized controller is shown in Fig. 13(a). The existing CIPN models for local controllers are shown in Fig. 13(b), (c).[11]

[11]Models of controllers A and B are identical; thus, only a single model of controller X for cylinder X is shown in Fig. 13(b), where $X \in \{A, B\}$.

Property	Average violation probability over scenarios (S1) Nominal conditions	(S2) Network unavail. due to congestion w/o app-level retries	(S2) Network unavail. due to congestion w/ app-level retries	(S3) Air pressure control fault
(P1)	0.0000	0.2690 [0.2459, 0.2908]	0.0000	0.0000
(P2)	0.0000	0.0330 [0.0237, 0.0423]	0.0000	0.0000
Property	Cylinder C average duty cycle [%]			
(P3)	6.1208 [6.1202, 6.1213]	—	6.0761 [6.0754, 6.0768]	5.6214 [5.6210, 5.6218]
Property	Inverse of average throughput [*seconds per unit*]			
(P4)	3.9368 [3.9366, 3.9370]	—	3.9671 [3.9668, 3.9674]	3.2154 [3.2152, 3.2156]

Table 2: Case Study I: Property violation probabilities and performance measures over different scenarios; for *network unavailability due to congestion without application-level retries*, performance measures are not given due to frequent operation interruptions. Confidence intervals are computed with 99% confidence.

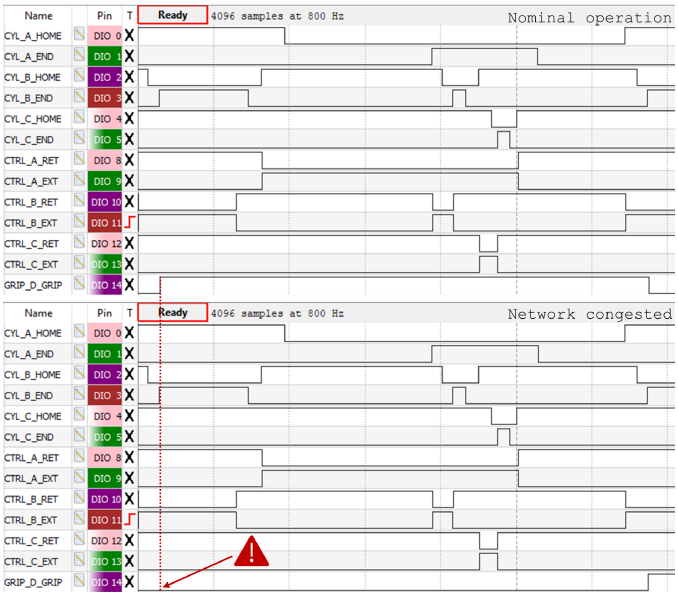


Figure 11: Case Study I: Experimental sensing/actuation timing diagrams: (top) correct execution under nominal conditions; (bottom) execution when the part is not gripped (see ALARM) due to a dropped packet from controller B to D, caused by network congestion.

This cylinder configuration is of special interest as cylinders A and B simultaneously stroke forward to load/unload parts; thus, controllers A and B in the distributed setting must simultaneously command the forward stroke of their respective cylinders, when commanded to do so by controller C (followed by completion of the rotary movement of cylinder C). While the communication from controller C to controllers A and B can be realized using a multicast protocol, guarantees must be provided that cylinders A and B both receive the signal, and synchronously issue actuation commands.

On the other hand, while broadcast is, multicast communication is not natively supported by the IEEE 802.15.4 low-power wireless communication standard employed in the previous case study. While there exist works on achieving multicast communication via IPv6 protocol (e.g., [19]), simultaneous delivery and processing of packets on multiple receiving nodes cannot be guaranteed. From

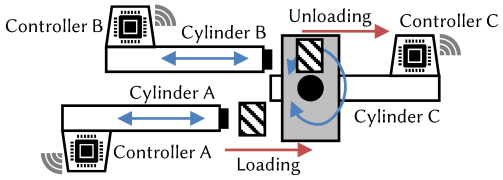


Figure 12: Case Study II: Industrial manipulator in a parallel loading/unloading configuration. Parts are loaded by cylinder A, and unloaded by cylinder B simultaneously, while the rotary cylinder C 'replaces' the loading/unloading positions by means of rotating the loading base.

the analysis of the SRN-models obtained as described in Sec. 3,[12] we found again that the translation of the controller communication semantics needed to be adjusted (from the one in Fig. 6(b)), as the resulted distributed automation system was violating safety properties. For instance, if only one of the controllers A or B receives the packet and commences actuation, the system transitions into an erroneous state; for example, a new part is loaded against the previous one that is not yet unloaded.

We thus modified the semantics of the communication-APIs in translation from the CIPNs as part of code generation; this was done to utilize the low-power, low-bandwidth synchronization protocol devised in [17], which is fully compatible with our LC platform implementation. When the synchronization protocol was employed, it enabled synchronous execution of actuation commands on controllers A and B, once both controllers received the signal from controller C. This may introduce additional delays, as additional handshaking between controllers A and B is added, as well as potential additional retransmissions between cylinders C and A, and C and B, which are originally nonexistent in the CIPN-based distributed automation model. Still, with (worst-case) 100 *ms* added to the production cycle time, sub-10 μs synchronization of cylinder A and B strokes is achieved, resulting in a verifiably safe system.

6 CONCLUSION

In this paper, we have presented a methodology that takes as inputs widely used CIPN-based controller models for industrial automation, which preclude correctness and fault-tolerance analysis of distributed automation deployments. Such models are transformed

[12] Complete model schematic is omitted due to space limitations.

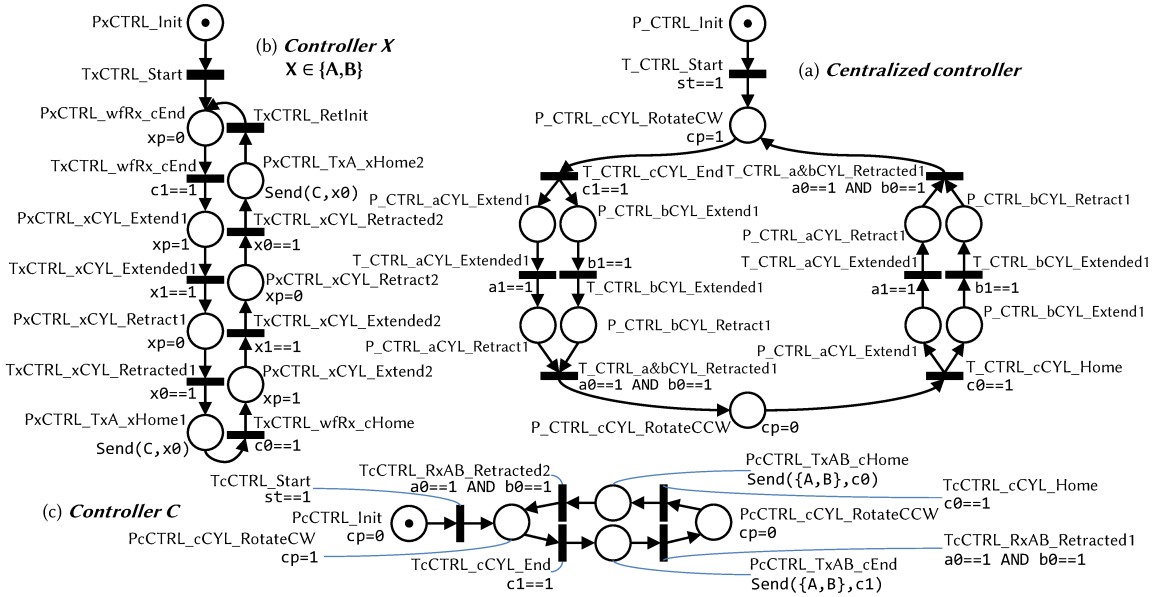


Figure 13: Case Study II — Global (a), and local (b),(c) CIPN controller models of the industrial manipulator from Fig. 12.

into an SRN-compatible representation that natively supports stochastic modeling of communication channels and system faults. We have shown how analysis/verification of the distributed automation system can be performed with probabilistic guarantees within the SRN paradigm. In addition, we have revised existing strategies for automatic synthesis of distributed control code based on results obtained from SRN models. To support dynamic deployment environments of IIoT-based automation systems, we have proven feasibility of an edge-based monitoring system that re-checks at runtime relevant system properties against models that are continuously updated from acquired real-time process and network channel measurements. We have evaluated our approach on real-world case-studies including industrial manipulators. As avenue for future work, we will explore inclusion of true non-determinism within the controller and channel models, which would allow us to capture adversarial affects on distributed automation systems.

ACKNOWLEDGMENTS

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 the Serbian Ministry of Education, Science and Technology research grant TR35004.

REFERENCES

[1] Peter Adolphs, Heinz Bedenbender, D Dirzus, M Ehlich, U Epple, M Hankel, R Heidel, M Hoffmeister, H Huhle, B Kärcher, et al. 2015. Reference architecture model industrie 4.0 (rami4. 0). *ZVEI and VDI, Status Report* (2015).

[2] Giacomo Bucci, Laura Carnevali, Lorenzo Ridi, and Enrico Vicario. 2010. Oris: a tool for modeling, verification and evaluation of real-time systems. *International Journal on Software Tools for Technology Transfer* 12, 5 (01 Sep 2010), 391–403.

[3] Franck Cassez and Olivier H. Roux. 2006. Structural translation from Time Petri Nets to Timed Automata. *J. of Systems and Software* 79, 10 (2006), 1456 – 1468.

[4] Gianfranco Ciardo, Jogesh K Muppala, and Kishor S Trivedi. 1992. Analyzing concurrent and fault-tolerant software using stochastic reward nets. *J. Parallel and Distrib. Comput.* 15, 3 (1992), 255–269.

[5] R. David and H. Alla. 2010. *Discrete, continuous, and hybrid petri nets (2nd edition).*

[6] Michel Diaz. 1982. Modeling and analysis of communication and cooperation protocols using Petri net based models. *Computer Networks* 6, 6 (1982), 419 – 441.

[7] S. Donatelli. 1993. Superposed stochastic automata: a class of stochastic Petri nets with parallel solution and distributed state space. *Performance Evaluation* 18, 1 (1993), 21 – 36.

[8] H. ElMaraghy, G. Schuh, W. ElMaraghy, F. Piller, P. Schönsleben, M. Tseng, and A. Bernard. 2013. Product variety management. *{CIRP} Annals - Manufacturing Technology* 62, 2 (2013), 629 – 652. https://doi.org/10.1016/j.cirp.2013.05.007

[9] P. Ferrari, A. Flammini, E. Sisinni, S. Rinaldi, D. Brandao, and M.S. Rocha. 2018. Delay Estimation of Industrial IoT Applications Based on Messaging Protocols. *IEEE Transactions on Instrumentation and Measurement* 67, 9 (2018), 2188–2199.

[10] Y. He, J. Guo, and X. Zheng. 2018. From Surveillance to Digital Twin: Challenges and Recent Advances of Signal Processing for Industrial Internet of Things. *IEEE Signal Processing Magazine* 35, 5 (2018), 120–129.

[11] Christophe Hirel, Bruno Tuffin, and Kishor S. Trivedi. 2000. SPNP: Stochastic Petri Nets. Version 6.0. In *Computer Performance Evaluation.Modelling Techniques and Tools*. Springer Berlin Heidelberg, 354–357.

[12] Z. Jakovljevic, V. Lesi, S. Mitrovic, and M. Pajic. 2018. Distributing Sequential Control for Manufacturing Automation Systems. *IEEE Transactions on Control Systems and Technology* (2018). submitted.

[13] Zivana Jakovljevic, Vidosav Majstorovic, Slavenko Stojadinovic, Srdjan Zivkovic, Nemanja Gligorijevic, and Miroslav Pajic. 2017. Cyber-Physical Manufacturing Systems (CPMS). In *Proceedings of 5th International Conference on Advanced Manufacturing Engineering and Technologies*. Springer International, 199–214.

[14] R Jayaparvathy, S Anand, S Dharmaraja, and S Srikanth. 2007. Performance analysis of IEEE 802.11 DCF with stochastic reward nets. *International Journal of Communication Systems* 20, 3 (2007), 273–296.

[15] Henning Kagermann, Johannes Helbig, Ariane Hellinger, and Wolfgang Wahlster. 2013. *Recommendations for implementing the strategic initiative INDUSTRIE 4.0.* Forschungsunion.

[16] Y. Koren, X. Gu, and W. Guo. 2018. Reconfigurable manufacturing systems: Principles, design, and future trends. *Frontiers of Mechanical Engineering* 13, 2 (2018), 121–136. https://doi.org/10.1007/s11465-018-0483-0

[17] V. Lesi, Z. Jakovljevic, and M. Pajic. 2016. Towards Plug-n-Play Numerical Control for Reconfigurable Manufacturing Systems. In *21st IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 1–8.

[18] Yue Ma, James J Han, and Kishor S Trivedi. 2001. Composite performance and availability analysis of wireless communication networks. *IEEE Transactions on Vehicular Technology* 50, 5 (2001), 1216–1223.

[19] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. 2007. *Transmission of IPv6 packets over IEEE 802.15.4 networks (No. RFC 4944).* Technical Report.

[20] T. Murata. 1989. Petri Nets: Properties, Analysis and Applications. *Proc. IEEE* 77, 4 (1989), 541–580.

[21] Srinivasan Ramani, Kishor S Trivedi, and Balakrishnan Dasarathy. 2000. Performance analysis of the CORBA event service using stochastic reward nets. In *Proc. of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS)*. 238–247.

[22] Kishor S Trivedi and Andrea Bobbio. 2017. *Reliability and Availability Engineering: Modeling, Analysis, and Applications.* Cambridge University Press.