CaaS: Enabling Control-as-a-Service for Time-Sensitive Networking

Zheng Yang*✉, Yi Zhao*, Fan Dang†, Xiaowu He*, Jiahang Wu*, Hao Cao*, Zeyu Wang* and Yunhao Liu†
* School of Software and BNRist, Tsinghua University † Global Innovation Exchange, Tsinghua University
{hmilyyz, zhaoyi.yuan31}@gmail.com, dangfan@tsinghua.edu.cn,
{horacehxw, jiahangok, i.haocao}@gmail.com, wzy20@mails.tsinghua.edu.cn, yunhaoliu@gmail.com

***Abstract*—Flexible manufacturing is one of the core goals of Industry 4.0 and brings new challenges to current industrial control systems. Our detailed field study on auto glass industry revealed that existing production lines are laborious to reconfigure, difficult to upscale, and costly to upgrade during production switching. Such inflexibility arises from the tight coupling of devices, controllers, and control tasks. In this work, we propose a new architecture for industrial control systems named Control-as-a-Service (CaaS). CaaS transfers and distributes control tasks from dedicated controllers into Time-Sensitive Networking (TSN) switches. By combining control and transmission functions in switches, CaaS virtualizes the industrial TSN network to one Programmable Logic Controller (PLC). We propose a set of techniques that realize end-to-end determinism for in-network industrial control and a joint task and traffic scheduling algorithm. We evaluate the performance of CaaS on testbeds based on real-world networked control systems. The results show that the idea of CaaS is feasible and effective, and CaaS achieves absolute packet delivery, 42-45% lower latency, and three orders of magnitude lower jitter. We believe CaaS is a meaningful step towards the distribution, virtualization, and servitization of industrial control.**

***Index Terms*—Time-Sensitive Networking, Industrial Network**

I. Introduction

Control systems are the brains of industrial automation. They underpin the success of modern industries and play a critical role in increasing production efficiency. Programmable Logic Controller (PLC) is the most common controller used in industrial systems [1], which has achieved huge success. Major PLC manufacturers include Siemens, Rockwell Automation, Mitsubishi Electric, *etc.* High-end PLCs are usually much more expensive than other computing devices with similar computation resources.

Recent years have witnessed the paradigm shift of industries towards Industry 4.0, also known as Industrial Internet, intelligent manufacturing, or new industrial revolution depending on the industrial upgrading policy of different countries. In this new paradigm, flexible manufacturing is among the core goals and brings new challenges to current PLC control systems. First, the production order patterns change greatly. Manufacturers are seeing more small production orders with more diverse specs. This forces the manufacturers to reconfigure production lines more frequently than ever before. Second, the number of connected devices continues to increase. In 2020, there are 17.7 billion connected industrial devices globally. It is estimated that this number will double by 2025, reaching 36.8 billion [2]. Last but not least, control tasks are evolving from simple relay logic to complex machine learning models. Tasks like defect detection have already benefited from computer vision to reduce operational costs and improve accuracy [3].

We conduct a field study on the auto glass industry, which is in urgent need of flexible manufacturing due to the changing order patterns from its customers. The limitations of current PLC control systems are three-fold: (1) **laborious to reconfigure a production line**. Sensor and actuator devices are hard-wired to PLC as discrete or analog IO. Engineers have to manually change the connections between devices and controllers and upload new control tasks to every PLC. (2) **difficult to upscale an industrial control system**. PLC works in a centralized way and every IO device needs to be directly connected to it. The number of connected devices is also constrained by the number of IO ports. (3) **costly to upgrade an existing control system**. Since tasks and controllers are tightly bound, when a PLC does not satisfy a new control task, workers have to stop the production and replace it with a more powerful but expensive one, which causes extra downtime and costs. More details are described in the next section. The root cause of the above limitations is the binding among devices, controllers, and tasks. This tight coupling creates difficulties in reconfiguration, upscaling, and upgrade of a control system.

In the past decade, new industrial communication technologies have been developing rapidly, such as PROFINET [4], POWERLINK [5], and Time-Sensitive Networking (TSN) [6]. Among them, TSN is widely recognized as the most promising technology for the next generation industrial networks. A number of major industrial technology companies are pushing the development of TSN technology, such as Cisco, Marvell, and NXP [7]. These technologies solve the problem of deterministic data transmission over Ethernet. Nevertheless, they do not aim at the flexibility issues of control systems discussed above.

In this work, we propose a new architecture for industrial control systems named Control-as-a-Service (CaaS), which transfers and distributes control tasks from dedicated controllers into network switches. By combining control and transmission functions in network switches, CaaS turns a network of switches into one virtual PLC, laying the foundations of the distribution, virtualization, and servitization of industrial

✉ Zheng Yang is the corresponding author.

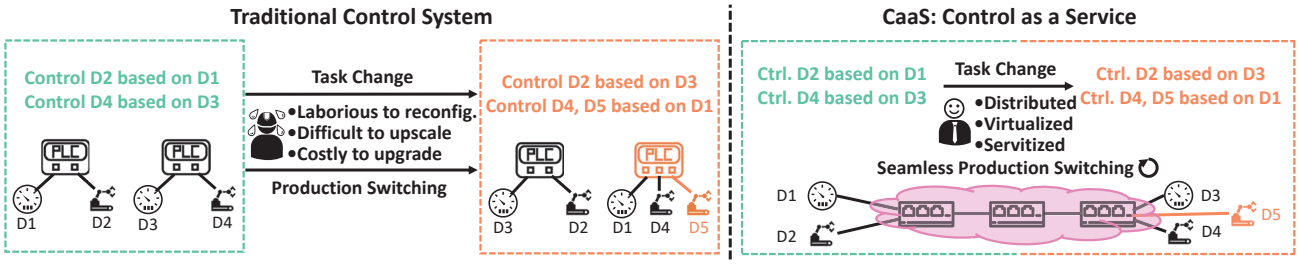


Figure 1: Traditional PLC-based control systems and Control-as-a-Service (CaaS) control systems.

control. In the design of CaaS, control is a service to be called and the details of the control mechanism are hidden from callers. For example, a device is "connected and controlled" without knowing where the controllers are; and a control task can run efficiently and deterministically without knowing in which controller and how it is executed. Fig. 1 compares traditional control systems and CaaS.

To implement CaaS, we are facing two main challenges: (1) *How to realize in-switch deterministic industrial control.* We propose a co-design of hardware and software for CaaS switch, implementing both data transmission and task execution functions. Specifically, we propose *Packetized PLC IO* to decouple the physical IO of devices and logical IO of controllers. Additionally, determinism is a fundamental requirement of industrial control. We present several techniques to ensure the end-to-end determinism, including *Dual DMA*, *Core Isolation*, and *Global-Time-Aware Execution*. (2) *How to schedule the data transmission and task execution in CaaS.* Prior work on industrial network scheduling only considers data transmission, assuming that the schedule of task execution is determined in advance. In CaaS, we propose *Task & Traffic Joint Scheduling Algorithm*, which models the relationship between tasks and traffic flows with our proposed flow-task dependency constraints.

The contributions of CaaS are as follows:

- To the best of our knowledge, CaaS is the first design that virtualizes the industrial network as one controller, which realizes the distribution, virtualization, and servitization of industrial control.
- We propose several techniques in the design and implementation of CaaS: (1) Packetized PLC IO, which eliminates the need to hard-wire IO devices to PLCs and lays the basis for the flexible arrangements of control tasks. (2) A set of techniques, which ensure end-to-end determinism, including both computation determinism and transmission determinism. (3) A joint scheduling algorithm, which for the first time considers the scheduling of task execution and traffic transmission at the same time.
- We evaluate the performance of CaaS on testbeds based on real-world networked control systems. The results show that the idea of CaaS is feasible and effective, and CaaS achieves absolute packet delivery, 42-45% lower latency, and three orders of magnitude lower jitter on all three testbeds.
- We make two contributions to the community: (1) We provide a qualitative and quantitative field study about the challenges of flexible production in a typical manufacturing industry, which helps to uncover new research issues on industrial networks for the community. (2) We make our CaaS implementation publicly available along with the Ziggo project[1]. CaaS can serve as a platform for the research about industrial control networks and also a toolkit for the implementation of deterministic systems.

In the rest of the paper, we introduce the motivating field study and some background in Sec. II. Then we overview CaaS in Sec. III and present its design in Sec. IV and Sec. V. After describing the implementation in Sec. VI, we evaluate the performance of CaaS in Sec. VII. At last, we discuss the related work in Sec. VIII and conclude the paper in Sec. IX.

II. Background and Motivation

A. Field Study

We begin with a field study on the auto glass industry. With the diversification and personalization of customer requirements, the auto glass industry is in urgent need of flexible manufacturing. We conducted a field study at one of the world's major auto glass factories (anonymity due to blind review). For the respondent factory, orders from automakers are shifting from large orders of a few specifications towards modest orders of various specifications. The CDF of each spec's production is depicted in Fig. 2. Over 79.7% are below 1,000 pieces. The change of order patterns urges the respondent company to embrace flexible manufacturing.

Specifically, we highlight three shortcomings of the current control system in light of flexible production.

Laborious to reconfigure: One production line has to prepare to manufacture more specs of glass than previously, necessitating frequent production switching. As summarized in Table I, the downtime of production switching results in 24% production reduction, the majority of which is attributable to the reconfiguration of control systems (*e.g.*, altering the connections between devices and PLCs and uploading new control tasks to PLCs) and pilot run.

Difficult to upscale: Due to the upgrade of their own products, carmakers often require the respondent factory to introduce new advanced inspections in glass production. As illustrated in Table II, six inspection devices were installed at various stages over the last few years. Installing new devices,

[1]Ziggo project: http://tns.thss.tsinghua.edu.cn/ziggo/

Table I: Production line switching.

Switch time (min)		Switch frequency	Production reduction
Change mold	Reconfig. & pilot run		
10	40	6-8/day	24%

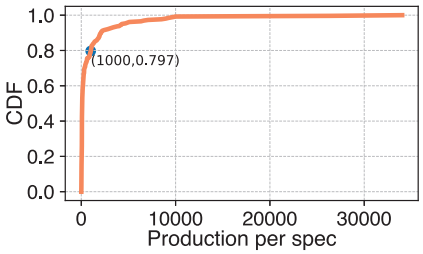


Figure 2: CDF of production per spec.

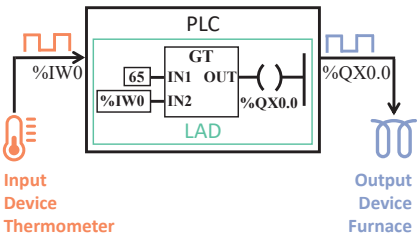


Figure 3: PLC and LAD.

Table II: Production line upgrade.

	Preprocessing	Printing	Quality control
# new devices	3	1	2
New tasks	Surface ins.[1] Edge ins. Size ins.	Printing ins.	Curvature ins. Appearance ins.
Computation	Arithmetic, CV, PID	CV	Arithmetic, CV

[1] inspection

however, is challenging for existing control systems. PLCs are centralized in their operation, and devices must be directly connected to them.

Costly to upgrade: As illustrated in the bottom row of Table II, the complexity of control tasks continues to increase as a result of technique advancements such as computer vision-based defect detection [3]. Due to the binding between tasks and controllers, PLCs that are unable to meet the requirements of the upgraded control tasks must be replaced with more powerful and expensive ones, despite the fact that some field-installed PLCs may have redundant computation resources.

The root cause of the aforementioned issues is the binding among devices, controllers, and tasks, which creates complications during reconfiguration, upscaling, or upgrade of an existing industrial control system.

B. PLC and LAD

Invented in the 1960s [8], PLC has grown to become the most widely used industrial control technology. A PLC executes control tasks periodically. Each execution cycle contains three stages: input scan, logic execution, and output writing. The primary programming language of PLCs is Ladder Logic (LAD), which resembles electrical diagrams and is defined in IEC 61131-3 [9]. Fig. 3 depicts a PLC running an LAD program. The PLC reads temperature data from the thermometer during each execution cycle. After that, a comparator circuit is used in the LAD program to compare the input value to 65 Celsius and determine the output value. Finally, the output value is written out to control the furnace's on/off operation.

C. TSN

TSN is a deterministic data transmission technique that is integrated into standard Ethernet. Fig. 4 illustrates two major

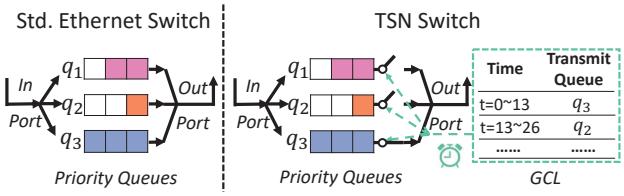


Figure 4: Standard Ethernet switch and TSN switch.

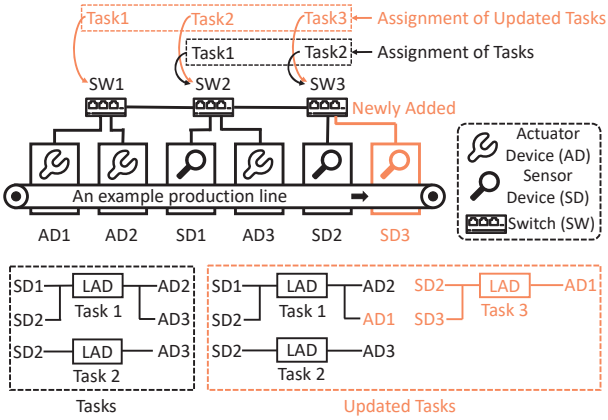


Figure 5: A working example of CaaS.

differences between an Ethernet switch and a TSN switch. First, TSN switches share a common sense of time, guaranteed by the time synchronization protocol defined by IEEE 802.1AS [10]. Second, standard Ethernet switches transmit data as long as no higher-priority packets are waiting. IEEE 802.1Qbv [11] enables TSN switches to reserve dedicated time slots for critical traffic, such as industrial control data. The reserved slots are specified by a schedule called the Gate Control List (GCL) contained within TSN switches. Centralized Network Configuration (CNC) is used to manage a TSN network, as defined in IEEE 802.1Qcc [12]. CNC generates a global schedule that specifies when data traffic should pass through each network link. Then CNC distributes the associated schedule, *i.e.*, GCL, to each TSN switch.

III. Overview

Due to the complexity of the design of CaaS, we first use a working example to demonstrate how CaaS works in Sec. III-A, putting aside technical details. Then we introduce the architecture of CaaS in Sec. III-B.

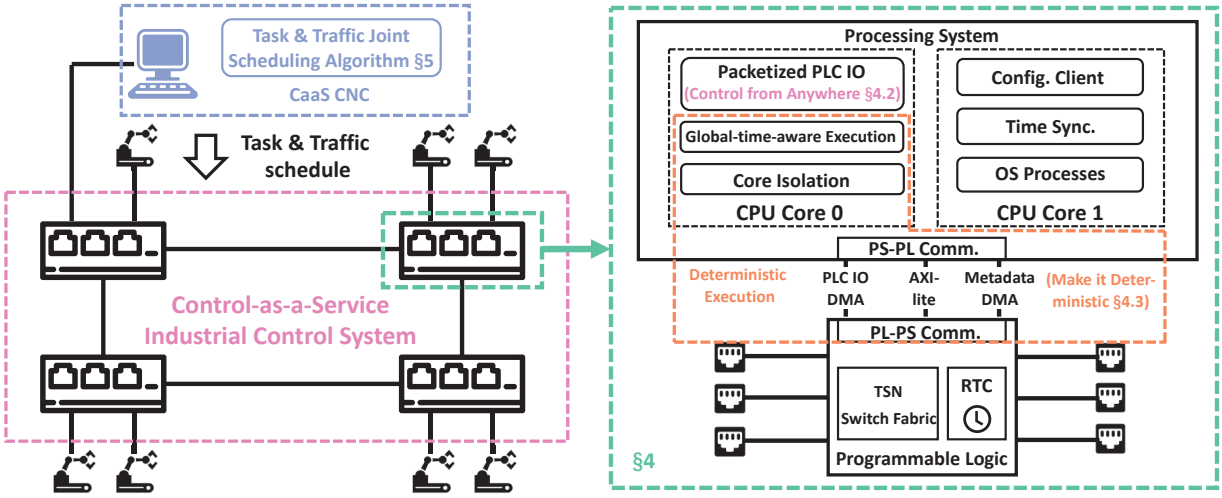


Figure 6: Overview of CaaS.

A. Working Example

As illustrated in Fig. 5, a typical industrial production line is based on a mechanical system that sequentially moves products from one device to the next, which may include both sensor and actuator devices. In contrast to traditional control systems, the CaaS system depicted in Fig. 5 makes use of CaaS switches to connect all devices into a single network that functions as a virtual PLC. Each network-connected device acts as an I/O device for this virtual PLC. Additionally, a CaaS CNC is connected to the network. Unlike TSN CNC, CaaS CNC not only schedules traffic transmission but also task execution. Engineers program CaaS CNC using ladder logic, just as they do with physical PLCs.

Initially, there are two tasks in this example as specified in Fig. 5. CaaS CNC schedules (1) where (in which PLC) and when tasks are executed, and (2) the corresponding data transmission over network links. Task 1 is assigned to CaaS switch SW2, while Task 2 is assigned to SW3. Simultaneously, the schedule for data transmission along the links is also synthesized and deployed to CaaS switches.

Additionally, Fig. 5 depicts a scenario in which a new sensor SD3 is integrated into the production line. A new task is also added as Task 3. At the same time, AD1 replaces AD3 as an output device of Task 1. Downtime is unavoidable during the reconfiguration of traditional production lines. Differently, CaaS can realize "connect and control" for newly added devices and seamless reconfiguration for updated control tasks. Then CaaS CNC recalculates the schedule for the updated tasks and instantly deploys it to the whole network.

B. System Architecture

As seen in Fig. 6, a CaaS industrial control system consists of devices, CaaS switches, and CNC. CaaS CNC runs the joint task and traffic scheduling algorithm, which will be presented in Sec. V.

We sketch the design of CaaS switches on the right of Fig. 6. CaaS switch adopts a hardware-software co-design to integrate data transmission and task execution. The hardware part is built upon an FPGA, *i.e.*, Programmable Logic (PL). The software part is built upon a dual-core CPU, *i.e.*, Processing System (PS).

PL implements functions that require high data rates or precise timing. Specifically, it implements the data transmission functions in accordance with IEEE TSN standards. PL also includes some components that realize 802.1AS time synchronization protocol, such as the real-time clock (RTC) and the module that timestamps incoming and outgoing packets.

PS implements functions that require complex processing. (1) The LAD control tasks assigned to the switch are executed by PS's CPU. Specifically, a *Packetized PLC IO* module (Sec. IV-B) is developed to decouple the physical IO of devices and the logical IO of LAD. We also propose several designs to ensure the time-sensitive execution of tasks (Sec. IV-C). (2) PS implements the logic processing and computation parts of the time synchronization protocol. (3) PS also receives and configures the issued schedules from CaaS CNC.

PS and PL communicate with three interfaces. The first is a DMA channel to transmit the IO data of control tasks. The second is another DMA channel, which transmits CaaS metadata, including both time synchronization data and schedule data. The last is an AXI-lite interface that writes configurations to or retrieves information from PL.

By leveraging the advantages of both PL and PS, CaaS realizes deterministic industrial control inside switches, which makes it possible to turn the whole network into one virtual PLC, and eventually enables the flexible reconfiguration, upscaling, and upgrade of a production line. The technical details will be introduced in the following sections.

IV. Switch Design

A. Switch Workflow

The workflow of CaaS switches is presented in Fig. 7. The packets arriving at a switch's PL originate from three sources (denoted by inward arrows to the switch fabric):

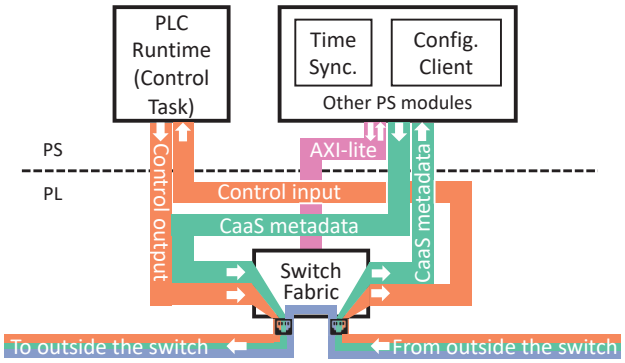


Figure 7: Workflow of CaaS switch

(1) From the outside of the switch: the packets sent by the connected devices and the packets forwarded by other switches; (2) Control output from PLC runtime: the packets carrying the control task's output; (3) CaaS metadata from PS: the packets carrying time synchronization and network configuration information.

The switch fabric will process these packets in accordance with their destination MAC addresses and EtherTypes. The switch fabric processes packets in three ways (denoted by outward arrows from the switch fabric): (1) To the outside of the switch: forwarding packets not destined for this switch to output ports directly by PL according to switch rules; (2) Control input: passing packets carrying control tasks' input data to PLC runtime; (3) CaaS metadata: passing packets carrying time synchronization and network configuration information to corresponding PS modules.

B. Control from Anywhere

A virtual PLC's high-level abstraction requires that control tasks can be executed at a low level on any CaaS switch within a network. We propose *Packetized PLC IO* to encapsulate LAD variables in Ethernet frames in order to decouple the physical connection between IO devices and PLCs.

First, let's see how IO is represented in PLC and LAD. As shown in the upper part of Fig. 8, the thermometer provides temperature as input, which is bound to the variable labeled as %IW0 in LAD. The furnace accepts an output to control its on/off, which is bound to the variable labeled as %QX0.0 (refer to [13] for the detailed naming conventions of variables). In every execution cycle, a PLC reads the inputs, executes the LAD program, and writes the outputs.

As a virtual PLC, the whole CaaS network has a unified address space to label the inputs and outputs of the connected devices. After the scheduling algorithm assigns the tasks to switches, CaaS CNC will configure the input devices of a task to encapsulate the input data and their address labels in an Ethernet frame and send it periodically. The destination MAC address will be set to the switch on which the associated task is scheduled. For example, the thermometer in the lower part of Fig. 8 encapsulates the temperature value (67) and the address label (%IW0) in an Ethernet frame and sends it to some switch in the network, which executes the control task.

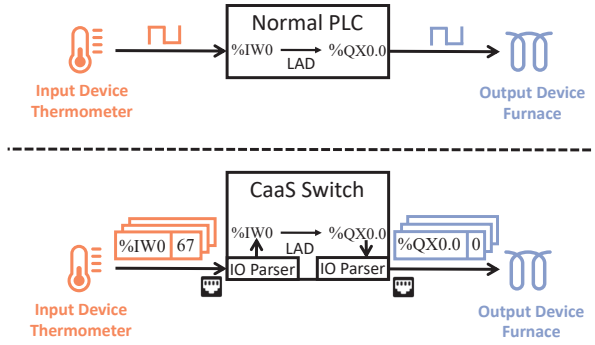


Figure 8: Wired PLC IO and Packetized PLC IO

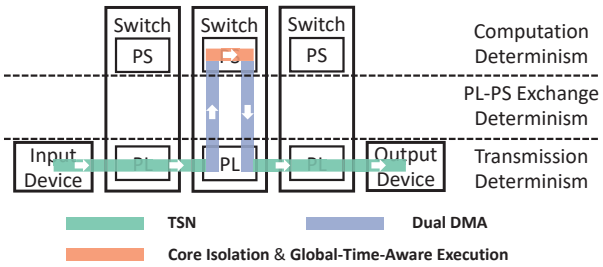


Figure 9: End-to-end determinism is guaranteed by TSN and three proposed designs.

After receiving the frame carrying input data, the *Packetized PLC IO* module binds the values to the LAD variables indicated by the addresses and triggers the next run of LAD programs. When the programs finish, the output value and labeling addresses (*e.g.*, 0 and %QX0.0 in Fig. 8) are embedded again in an Ethernet frame and sent to the relevant output devices (*e.g.*, the furnace in Fig. 8). For devices that do not support TSN, a converter or an encapsulator may be needed.

C. Make it Deterministic

Deterministic real-time control is essential to industrial systems. *How to ensure the end-to-end determinism* is the second challenge in the design of CaaS switch. The end-to-end determinism consists of three parts: the transmission determinism of the networks, the determinism of PL-PS data exchange, and the computation determinism of the controllers.

In Fig. 9, we analyze the stages of the control process and show how TSN and the three proposed designs guarantee the end-to-end determinism. As shown in the figure, a packet carrying input data is sent to the switch that executes the control task. Then the results are packetized and sent to the output device. In this process, TSN guarantees that the time when the packet arrives at PL's switch fabric is deterministic. The first proposed design, *Dual DMA*, provides determinism in the data exchange from PL to PS. The second design, *Core Isolation*, ensures that the computation interval of control tasks in PS is deterministic. The third design, *Global-Time-Aware (GTA) Execution*, guarantees the determinism of when the control task starts execution. At last, the output data is embedded in packets and passed from PS to PL again via

Dual DMA, after which the TSN network delivers the data to the output device. Next, we explain these three designs respectively.

Dual DMA: Two kinds of data are exchanged between PS and PL, *i.e.*,CaaS metadata and control tasks' IO data. Control tasks' IO data are time-sensitive and need deterministic exchange between PL and PS, while the CaaS metadata are not. To protect the transmission of time-sensitive data from that of non-time-sensitive data, we design two DMA between PL and PS. One DMA transmits control data and the other transmits CaaS metadata.

Core Isolation: Besides PLC runtime, there are a number of other processes running in PS at the same time, including configuration client, time synchronization, and other switch functions. To avoid the influence of these processes on PLC runtime, we reserve a dedicated CPU core for the execution of control tasks. All other processes will only run on the other CPU core.

GTA Execution: The PLC runtime we use is based on OpenPLC [14], which launches the execution according to CPU time, instead of the global synced time. So we propose *GTA Execution*. Instead of CPU time, the PLC runtime of CaaS operates based on the synced time retrieved from PL's RTC module via the AXI-lite interface. Then it launches the control tasks at the specified time in every period.

V. Joint Scheduling Algorithm

The global schedule calculated by CaaS CNC should specify the assignments of tasks to CaaS switches, as well as the start time and finish time of the control tasks' execution. Each task is related to several data flows, including flows from input devices to controller switches and from controller switches to output devices. The schedule also indicates when these flows pass each link along their paths, so the switches can reserve time slots in advance.

A. Input Notation

We define the scheduling problem as finding a valid schedule so that all tasks' latency requirements are satisfied. The input of the scheduling problem includes the network topology G and the set of tasks J. The topology is modeled as a directed graph $G = (V, E)$. V is the set of vertices, representing switches and devices. E is the set of edges, representing the network links. If two nodes s and e are linked, two edges, $\langle s, e\rangle$ and $\langle e, s\rangle$, are added to E since the link is full-duplex. A switch node $v \in V$ has an attribute $v.d$, which is the switch's forwarding delay. It is the minimum offset between the time slots reserved on consecutive links. As for the input tasks, a task $j_i \in J$ is characterized by 5 attributes: $(j_i.S, j_i.D, j_i.T, j_i.P, j_i.MD)$. S is the set of input devices and D is the set of output devices. The information about input data length and output data length is also included. T is the execution time of the task and P is its period. MD is the maximum allowed delay of the task.

B. Formulation

We model the joint scheduling problem as a Satisfiability Modulo Theory (SMT) problem. It then can be solved by an SMT solver.

Task Constraints: For each task j_i, we define three scheduling variables: $(j_i.host, j_i.start, j_i.o)$, where $j_i.host \in V$ is the switch that executes this task, $j_i.start$ is the start time of execution and $j_i.o$ indicates if the task is scheduled to start in the next period. The constraint for tasks is that the tasks assigned to the same host cannot overlap with each other in time dimension.

Flow Constraints: A task j_i is related to $|j_i.S|$ input data flows and $|j_i.D|$ output data flows. The set of all data flows related to j_i is represented by F_i. We define a series of indicator variables to describe the flows' usage of links as in [15]:

$$x_{ikse} = \begin{cases} 1, & \text{if flow } f_{ik} \text{ passes link } (s, e), \\ 0, & \text{if flow } f_{ik} \text{ does not pass link } (s, e), \end{cases}$$

where $f_{ik} \in F_i$ is the kth flow related to j_i and $(s, e) \in E$ represents a link. For input flows, $f_{ik}.src \in j_i.S$ is the source device. For output flows, $f_{ik}.dst \in j_i.D$ is the destination device. We define scheduling variables t_{ikse} and o_{ikse} to describe the reserved time slots on links for data.

- t_{ikse} is an integer variable in range $[0, j_i.P)$, which is the start time of f_{ik}'s time slot on link (s, e).
- o_{ikse} is 0 or 1, indicating if the scheduled time slots are in the next period.

When a flow passes network links, the scheduled slots on successive links cannot precede those on former links. Additionally, the time slots reserved for different flows on the same link cannot overlap with each other.

Flow-Task Dependency Constraints: The CaaS switches where the tasks are executed determine the destinations of input data flows and the sources of output data flows. We assume the flows always follow the shortest paths between sources and destinations. Thus, the flow path is also determined. This dependency is described by the following constraints. V_{sw} represents the set of CaaS switches. $shortestpath$ finds the shortest path between two nodes:

$$\begin{aligned} &\forall j_i \in J, \forall f_{ik} \in F_i, \forall h \in V_{sw} : \\ &m \leftarrow \begin{cases} f_{ik}.src, & \text{if } f_{ik} \text{ is an input flow}, \\ f_{ik}.dst, & \text{if } f_{ik} \text{ is an output flow}, \end{cases} \\ &sp \leftarrow shortestpath(h, m). \\ &\forall (s, e) \in E, \\ &\quad AddConstraints(Imply(j_i.host = h \wedge (s, e) \in sp, x_{ikse} = 1)), \\ &\quad AddConstraints(Imply(j_i.host = h \wedge (s, e) \notin sp, x_{ikse} = 0)), \end{aligned} \tag{1}$$

where $AddConstraints$ adds the constraints to the SMT model, and $Imply$ is a type of SMT constraint, which means that the first argument implies the second argument. We set

t_{ikse} and o_{ikse} to 0 for the links that are not on the path of the flow:

$$\begin{aligned}&\forall j_i \in J, \forall f_{ik} \in F_i, \forall (s,e) \in E : \\ &\quad AddConstraints(Imply(x_{ikse} = 0, t_{ikse} = 0 \wedge o_{ikse} = 0)).\end{aligned} \tag{2}$$

Additionally, the scheduled interval for task execution should be after the arrival of input data. F_i^{src} represents the input flows of j_i and $f_{ik}.l$ represents the time needed to transmit the input or output data:

$$\begin{aligned}&\forall j_i \in J, \forall f_{ik} \in F_i^{src} : \\ &endt \leftarrow \sum_{(s,e)\in E} If(x_{ikse} = 1 \wedge j_i.host = e, \\ &\qquad\qquad t_{ikse} + o_{ikse} \times j_i.P + f_{ik}.l, 0), \\ &AddConstraints(j_i.start + j_i.o \times j_i.P \geq endt),\end{aligned} \tag{3}$$

where $If(p, v_1, v_2)$ is an SMT function. Its value equals v_1 if p is true, otherwise its value equals v_2. The destinations of input flows are uncertain. Thus the arrival time of input data cannot be directly represented by scheduling variables t_{ikse} and o_{ikse}.

Similarly, the scheduled transmission of output data should also be after the completion of control tasks. F_i^{dst} represents the output flows of j_i:

$$\begin{aligned}&\forall j_i \in J, \forall f_{ik} \in F_i^{dst} : \\ &startt \leftarrow \sum_{(s,e)\in E} If(x_{ikse} = 1 \wedge j_i.host = s, \\ &\qquad\qquad t_{ikse} + o_{ikse} \times j_i.P, 0), \\ &AddConstraints(j_i.start + j_i.o \times j_i.P + \\ &\qquad\qquad j_i.T \leq startt).\end{aligned} \tag{4}$$

At last, the schedule must ensure that the latency of the task, including both transmission latency and computation latency, does not exceed the maximum allowed value. The latency is defined as the interval between the arrival of the last output flow and the departure of the first input flow:

$$\begin{aligned}&\forall j_i \in J : \\ &srct_i \leftarrow min(\{ \sum_{(s,e)\in E, s=f_{ik}.src} (t_{ikse} + o_{ikse} \times j_i.P) \\ &\qquad\qquad |f_{ik} \in F_i^{src}\}), \\ &dstt_i \leftarrow max(\{ \sum_{(s,e)\in E, e=f_{ik}.dst} (t_{ikse} + o_{ikse} \times j_i.P) \\ &\qquad\qquad + f_{ik}.l | f_{ik} \in F_i^{dst}\}), \\ &AddConstraints(dstt_i - srct_i \leq j_i.MD),\end{aligned} \tag{5}$$

Optimization Objective: An optional optimization objective can be added to the model to minimize the tasks' latency, instead of just finding a valid schedule:

$$Minimize(\sum_{j_i \in J} dstt_i - srct_i), \tag{6}$$

where $dstt$ and $srct$ are defined in Eq.(5).

VI. Implementation

We implement CaaS switches on Xilinx ZYNQ-7000 SoC [16]. ZYNQ-7000 SoC combines the hardware programmability of an FPGA and the software programmability of an ARM-based processor, which matches the architecture design of CaaS switches.

TSN Switch Fabric: TSN switch fabric in PL is implemented in Verilog following IEEE TSN standards [11], [12], [17], [18]. Each port has three frame queues in the switch fabric, one for the I/O data of control tasks, one for CaaS metadata, and one for background traffic.

Time Synchronization: We implement the time synchronization protocol defined in IEEE 802.1AS [10]. The modules that require precise timing including real-time clock and timestamping module are implemented using Verilog and run on PL. The modules implementing the synchronization algorithm of 802.1AS are developed in C/C++ and run on PS.

CaaS PLC Runtime: The PLC runtime of CaaS is developed based on an open-source project: OpenPLC [14]. We replace the hardware IO layer of OpenPLC with our *Packetized PLC IO* layer. We also connect PL's real-time clock module to PS through the AXI-lite interface and implement a Linux driver to access the synced time. This is the basis to implement *Global-Time-Aware Execution*.

VII. Evaluation

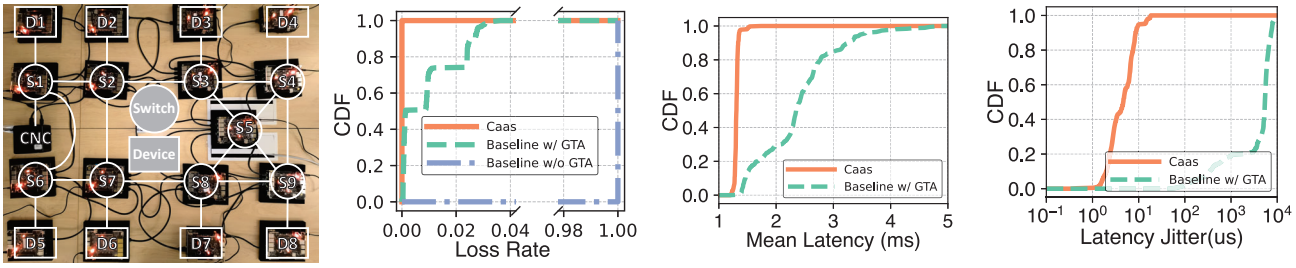
A. Overall System Performance

In this section, we evaluate the overall system performance of CaaS on two testbeds.

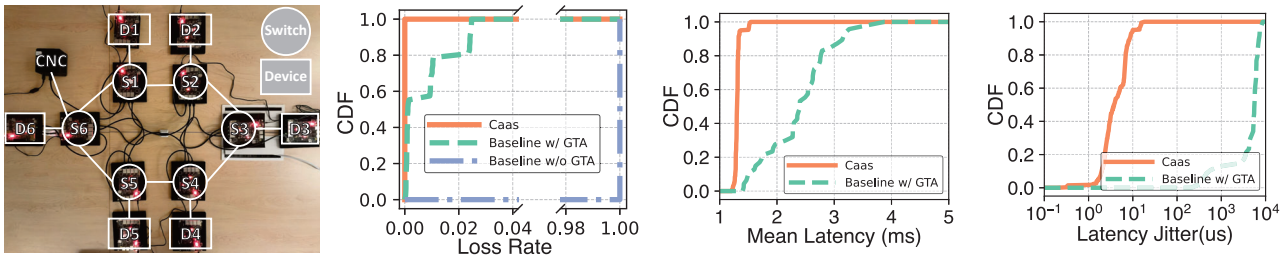
1) Setup: **Topology:** The first testbed A380 uses the topology shown in Fig. 10a, which is a simplified version of the control network used on Airbus A380 [19]. It consists of 9 switches and 8 devices. The second testbed Ring6 uses the ring topology shown in Fig. 10e, consisting of 6 switches and 6 devices. Ring topology is also common in industrial control networks [20], [21].

Baseline: The goal of CaaS is to realize a flexible industrial control system that can satisfy the determinism requirements of critical control tasks. It is very difficult if not impossible to set up a baseline with commodity PLCs and industrial switches since their systems are closed and proprietary. So we compare CaaS with two baselines built upon TSN and OpenPLC. To make baselines runnable on testbeds, both of them implement *Packetized PLC IO*. *Baseline w/o GTA* is a vanilla solution that has single DMA, no Core Isolation, and no GTA Execution. *Baseline w/ GTA* implements GTA Execution additionally so it can follow CNC's task execution schedule. Two baselines adopt the two-step scheduling algorithm, which first schedules tasks under *Task Constraints* described in Sec. V-B, and then schedules network traffic under *Flow Constraints* and *Flow-Task Dependency Constraints*.

Setting: We conduct 50 experiments on each testbed. In each experiment, we set the period of tasks as $33ms$, the task execution time as $1ms$, which are typical values in industrial systems [22], [23]. We randomly generate n tasks with m



(a) A380 network topology (b) A380, CDF of loss rate (c) A380, CDF of mean latency (d) A380, CDF of jitter



(e) Ring6 network topology (f) Ring6, CDF of loss rate (g) Ring6, CDF of mean latency (h) Ring6, CDF of jitter

Figure 10: Overall performance on different testbeds.

sensor devices and q actuator devices, where n equals the number of CaaS switches, m and q are randomly chosen from 1 to 4 [19]. Both scheduling algorithms are set to minimize the tasks' latency.

Metric: For each task, we measure the average latency, jitter (standard deviation of latency), and packet loss rate for 1,000 periods. A packet is dropped if the relevant task misses the scheduled computation time.

2) Results: Fig. 10 shows results on different testbeds. Overall, CaaS achieves absolute packet delivery, 42-45% lower latency, and three orders of magnitude lower jitter on both testbeds.

Fig. 10b and 10f display the loss rate results. On both testbeds, the loss rate of *Baseline w/o GTA* equals 1, *i.e.*, no packet can be transmitted as scheduled because the PLC runtime is unaware of the global synced time. Thus we introduce the second baseline, *Baseline w/ GTA*. The CDFs show that CaaS achieves 0 packet loss on both testbeds, while for *Baseline w/ GTA*, the 99 percentile loss rate is 3.18% (A380) and 2.47% (Ring6).

The distribution of mean latency is shown in Fig. 10c and 10g. It is clear that CaaS shows more concentrated latency distribution and lower average latency than baseline. Taking Fig. 10c as an example, most tasks' latency in CaaS lies in a narrow range around $1.31ms$, while the latency of *Baseline w/ GTA* is distributed evenly from $1.46ms$ to $3.24ms$. On average, CaaS achieves 42-45% lower latency. There are two reasons for the improvement. First, *Task & Traffic Joint Scheduling Algorithm* can optimize the schedule globally by tuning the task schedule and traffic schedule at the same time, while the two-step scheduling method can only find a local minimum. Second, techniques in Sec. IV-C ensure the determinism of PL-PS communication and task execution. Therefore, CaaS can follow the schedule perfectly, while the baseline always deviates from the plan.

As shown in Fig. 10d and 10h, the jitter of CaaS is almost negligible compared to the baseline. Specifically, the median jitter on A380 and Ring6 is $4.6\mu s$, $3.7\mu s$ for CaaS, and $5.4ms$, $5.6ms$ for baseline, respectively. The jitter of CaaS is over three orders of magnitude lower than baseline. The main reason is the absence of *Core Isolation* in the baseline, which causes uncertain execution time.

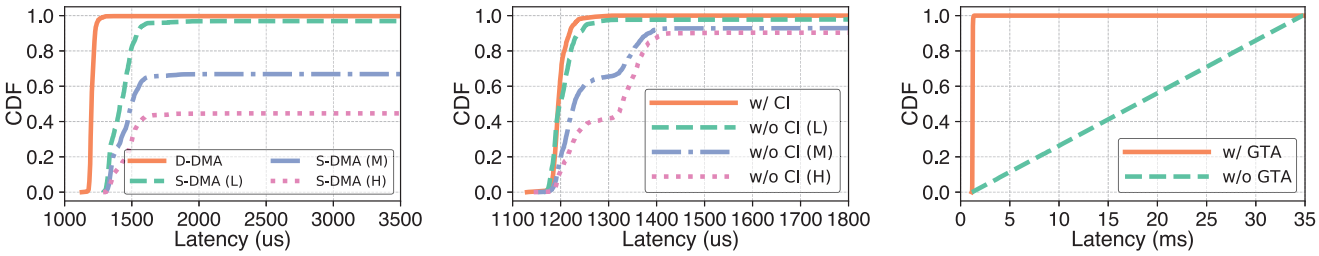
B. Component Study of CaaS Switch

In this section, we evaluate the key components in the design of CaaS switch, including *Dual DMA*, *Core Isolation*, and *Global-Time-Aware Execution*.

1) Setup: We set up a mini testbed for the component study, which consists of two devices connected by one CaaS switch. A control task is executed on the switch, with one device as its sensor and the other as its actuator. The task's period and execution time are $33ms$ and $1ms$, respectively. We compare CaaS with its three variants to find out the effectiveness of each component of CaaS switch: (1) *CaaS w/o Dual DMA*. Both control data and CaaS metadata share a single DMA. (2) *CaaS w/o Core Isolation.* Both CPU cores are available for all processes. (3) *CaaS w/o GTA Execution.* The PLC runtime scans DMA input based on CPU time, instead of the global synced time.

2) Results: In each experiment, the task is executed 1,000 times and the task's latency is recorded. The CDF of each method's latency under different settings is shown in Fig. 11. The lost packets' latency is seen as infinity.

Dual DMA: In the first experiment, we evaluate CaaS's performance w/ and w/o Dual DMA (*D-DMA* and *S-DMA*). The results are shown in Fig. 11a. We also study the impacts of traffic load on the two methods. The results show that



(a) CaaS w/ Dual (D-) and w/ Single (S-) DMA (b) CaaS w/ and w/o Core Isolation (CI) (c) CaaS w/ and w/o GTA Execution

Figure 11: CDF of latency for component study. (a) Light (L), medium (M) and heavy (H) refer to network loads of 0Mbps, 5Mbps and 10Mbps, respectively, consumed by extra data transmitted from PL to PS. (b) L, M and H refer to 0%, 50% and 75% CPU usage, consumed by background processes.

the performance of *D-DMA* is almost identical from light to heavy traffic load. Thus only one CDF of *D-DMA* is drawn in Fig. 11a due to space considerations. However, *S-DMA* suffers from heavier traffic load. The packet loss rate rises from 3.1% to 55.4%. For *D-DMA*, PLC IO data are transmitted through a dedicated DMA, thus the loss rate is always 0, not affected by other traffic.

Core Isolation: In the second experiment, we evaluate CaaS's performance w/ and w/o Core Isolation (*w/ CI* and *w/o CI*). The results are shown in Fig. 11b. We also study the impacts of CPU usage on the two methods. Since processes other than PLC runtime cannot run on the isolated CPU core, the performance of *w/ CI* is not affected by CPU usage. Again, only one CDF line of *w/ CI* is reserved for simplicity. In contrast, the performance of *w/o CI* degrades significantly with CPU usage increasing. The uncertainty of latency is due to the influence of other processes on the execution of control tasks. Furthermore, the uncertainty of latency also causes some tasks to violate the execution schedule, resulting in packet loss.

GTA Execution: In the third experiment, we evaluate CaaS's performance w/ and w/o GTA Execution (*w/ GTA* and *w/o GTA*). Different from previous experiments that drop the packets if the relevant tasks violate the execution schedule, we reserve all packets in this experiment to study the detailed impacts of GTA Execution. The results are shown in Fig. 11c. The latency of *w/ GTA* is very stable around $1.2ms$, while the latency of *w/o GTA* is scattered evenly in one period (from about $1ms$ to $35ms$). This is because *w/o GTA* executes control tasks according to its local CPU clock, which drifts away from the synced global time.

VIII. Related Work

Virtual PLC: In recent years, some work tries to replace the hardware PLCs in industries with virtual ones running in virtual machines or containers. For example, Givehchi *et al.* [24] provide a case study to build virtual PLCs in cloud servers in order to bring more agility to industrial automation systems. Hegazy *et al.* [25] also deploy the control tasks to cloud servers and propose an adaptive delay compensator and a distributed fault tolerance method to improve timeliness and reliability. Due to the network delay and jitter of cloud servers, the determinism of virtual PLCs is limited even with private clouds. They can only satisfy the requirements of soft real-time applications [24]. As a result, critical control tasks still rely on hardware PLCs. We need a new design for industrial control systems that can both provide agility and satisfy the determinism requirements of critical control tasks.

TSN Scheduling: The scheduling of TSN traffic has been studied widely. Most of them treat TSN scheduling as one time offline problem. Craciunas *et al.* first formulate TSN scheduling as Satisfiability Modulo Theories (SMT) problem in 2016 [26], based on their previous work on other deterministic networks [27]–[29]. Besides, some work uses Integer Linear Programs (ILP) to formulate and solve the problem in different ways [30]–[32]. Another group of work considers the dynamic change of network topology or traffic flows, and thus focuses on accelerating the scheduling algorithms. They design heuristics like Tabu search [33]–[35], incremental backtracking [36], greedy randomized search [37], [38], or deep reinforcement learning [39] to increase the number of flows that can be scheduled under a fixed time budget. In a nutshell, prior work only focuses on traffic scheduling, while CaaS jointly considers task and traffic scheduling problems.

IX. Conclusion

In this work, we propose CaaS, a new architecture for industrial control systems. It transfers and distributes control tasks from dedicated controllers into network switches. In the design of CaaS, control is a service and the details of the control mechanism are hidden from callers. The evaluation results show that CaaS is feasible and effective, and achieves significant performance gain. CaaS, we believe, is a significant step towards the distribution, virtualization, and servitization of industrial control. Numerous industrial control system challenges must be addressed from a network perspective. The authors have provided public access to their code at https://doi.org/10.5281/zenodo.7489411.

Acknowledgment

This work is supported in part by the National Key Research Plan under grant No. 2021YFB2900100, the NSFC under grant No. 61832010, 62202263, 62202262, and 61972131.

REFERENCES

[1] WikiPedia. (2022) Programmable logic controller. [Online]. Available: https://en.wikipedia.org/wiki/Programmable_logic_controller

[2] TechTarget. (2022) Industrial iot connections to reach 37 billion by 2025. [Online]. Available: https://www.computerweekly.com/news/252491495/Industrial-IoT-connections-to-reach-37-billion-by-2025

[3] H. Dong, K. Song, Y. He, J. Xu, Y. Yan, and Q. Meng, "Pga-net: Pyramid feature fusion and global context attention network for automated surface defect detection," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 12, pp. 7448–7458, 2020.

[4] WikiPedia. (2021) Profinet. [Online]. Available: https://en.wikipedia.org/wiki/Profinet

[5] ——. (2021) Ethernet powerlink. [Online]. Available: https://en.wikipedia.org/wiki/Ethernet_Powerlink

[6] ——. (2021) Time-sensitive networking. [Online]. Available: https://en.wikipedia.org/wiki/Time-Sensitive_Networking

[7] M. Research. (2021) Time-sensitive networking market. [Online]. Available: https://www.marketsandmarkets.com/Market-Reports/time-sensitive-networking-market-215000493.html

[8] Polycase. (2022) What is a programmable logic controller (plc). [Online]. Available: https://www.polycase.com/techtalk/electronics-tips/what-is-a-programmable-logic-controller.html

[9] *Programmable controllers - Part 3: Programming languages*, IEC Std. 61 131-3:2013, 2013.

[10] *Timing and Synchronization for Time-Sensitive Applications*, IEEE Std. 802.1AS, 2020.

[11] *Enhancements for Scheduled Traffic*, IEEE Std. 802.1Qbv, 2015.

[12] *Stream Reservation Protocols (SRP) Enhancements and Performance Improvements*, IEEE Std. 802.1Qcc, 2018.

[13] OpenPLC. (2021). [Online]. Available: https://www.openplcproject.com/reference/plc-addressing/

[14] T. R. Alves, M. Buratto, F. M. de Souza, and T. V. Rodrigues, "Openplc: An open source alternative to automation," in *IEEE Global Humanitarian Technology Conference (GHTC 2014)*, 2014, pp. 585–589.

[15] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl, "Ilp-based joint routing and scheduling for time-triggered networks," in *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, ser. RTNS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 8–17. [Online]. Available: https://doi.org/10.1145/3139258.3139289

[16] Xilinx. (2021) Socs with hardware and software programmability. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[17] *Bridges and Bridged Networks*, IEEE Std. 802.1Q, 2018.

[18] *Forwarding and Queuing Enhancements for Time-Sensitive Streams*, IEEE Std. 802.1Qav, 2009.

[19] F. Boulanger, D. Marcadet, M. Rayrole, S. Taha, and B. Valiron, "A time synchronization protocol for a664-p7," in *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*. IEEE, 2018, pp. 1–9.

[20] P. N. America. (2022) Industrial topology options and profinet. [Online]. Available: https://us.profinet.com/wp-content/uploads/2019/08/Topology.pdf

[21] W. Voss. (2019) Industrial ethernet guide - network topologies. [Online]. Available: https://copperhilltech.com/blog/industrial-ethernet-guide-network-topologies/

[22] PLC-City. (2021) Siemens simatic s7-1500. [Online]. Available: https://www.plc-city.com/shop/en/siemens-simatic-s7-1500.html

[23] P. N. e. V. (PNO). (2022) Profinet technology and application - system description. [Online]. Available: https://www.profibus.com/download/profinet-technology-and-application-system-description

[24] O. Givehchi, J. Imtiaz, H. Trsek, and J. Jasperneite, "Control-as-a-service from the cloud: A case study for using virtualized plcs," in *2014 10th IEEE Workshop on Factory Communication Systems (WFCS 2014)*. IEEE, 2014, pp. 1–4.

[25] T. Hegazy and M. Hefeeda, "Industrial automation as a cloud service," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 10, pp. 2750–2763, 2014.

[26] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS '16. New York, NY, USA: Association for Computing Machinery, Oct. 2016, pp. 183–192.

[27] F. Pozo, G. Rodriguez-Navas, H. Hansson, and W. Steiner, "Smt-based synthesis of ttethernet schedules: A performance study," in *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*. IEEE, 2015, pp. 1–4.

[28] F. Pozo, W. Steiner, G. Rodriguez-Navas, and H. Hansson, "A decomposition approach for smt-based schedule synthesis for time-triggered networks," in *2015 IEEE 20th conference on emerging technologies & factory automation (ETFA)*. IEEE, 2015, pp. 1–8.

[29] S. S. Craciunas and R. S. Oliver, "Combined task-and network-level scheduling for distributed time-triggered systems," *Real-Time Systems*, vol. 52, no. 2, pp. 161–200, 2016.

[30] P. Pop, M. L. Raagaard, M. Gutierrez, and W. Steiner, "Enabling fog computing for industrial automation through time-sensitive networking (tsn)," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 55–61, 2018.

[31] J. Falk, F. Dürr, and K. Rothermel, "Exploring practical limitations of joint routing and scheduling for tsn with ilp," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2018, pp. 136–146.

[32] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl, "Ilp-based joint routing and scheduling for time-triggered networks," in *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, 2017, pp. 8–17.

[33] F. Glover and M. Laguna, "Tabu search," in *Handbook of combinatorial optimization*. Springer, 1998, pp. 2093–2229.

[34] F. Dürr and N. G. Nayak, "No-wait packet scheduling for ieee time-sensitive networks (tsn)," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 203–212. [Online]. Available: https://doi.org/10.1145/2997465.2997494

[35] J. Yan, W. Quan, X. Jiang, and Z. Sun, "Injection time planning: Making cqf practical in time-sensitive networking," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 616–625.

[36] W. Steiner, "An evaluation of smt-based schedule synthesis for time-triggered multi-hop networks," in *2010 31st IEEE Real-Time Systems Symposium*, 2010, pp. 375–384.

[37] M. G. Resende and C. C. Ribeiro, "Grasp: Greedy randomized adaptive search procedures," in *Search methodologies*. Springer, 2014, pp. 287–312.

[38] V. Gavriluţ and P. Pop, "Scheduling in time sensitive networks (tsn) for mixed-criticality industrial applications," in *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, 2018, pp. 1–4.

[39] C. Zhong, H. Jia, H. Wan, and X. Zhao, "DRLS: A Deep Reinforcement Learning Based Scheduler for Time-Triggered Ethernet," in *2021 International Conference on Computer Communications and Networks (ICCCN)*, Jul. 2021, pp. 1–11.