CaaS: Enabling Control-as-a-Service for Real-Time Industrial Networking

Zheng Yang*, *Fellow, IEEE,* Zeyu Wang, Xiaowu He, Yi Zhao, *Student Member, IEEE,* Fan Dang, *Senior Member, IEEE,* Jiahang Wu, Hao Cao, *Student Member, IEEE,* Yunhao Liu, *Fellow, IEEE*, Qiang Ma, *Member, IEEE*

***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 network switches. By combining control and transmission functions in switches, CaaS virtualizes the whole industrial 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. To sum up, the root cause of the aforementioned issues lies in the tight binding among devices, controllers, and tasks in existing industrial control systems. It yields significant complications during reconfiguration, upscaling, or upgrade of an existing industrial control system.

During the past decade, new industrial communication technologies have been developing rapidly, *e.g.*, PROFINET [4], POWERLINK [5], and Time-Sensitive Networking (TSN) [6]–[9]. These technologies enables deterministic data transmission over Ethernet, improving the bandwidth, latency, and compatibility of industrial control networks. Nevertheless, they are designed as part of traditional industrial control systems, and still rely on dedicated controllers like PLC to execute the control logic. Thus, these technologies alone are not able to solve the flexibility issues 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 control. In the design of CaaS, control is a service to be called and the details of the control mechanism are hidden from

A preliminary version of this article appeared in the IEEE International Conference on Computer Communications (IEEE INFOCOM 2023).

Zheng Yang, Zeyu Wang, Xiaowu He, Yi Zhao, Jiahang Wu and Hao Cao are with the School of Software and BNRist, Tsinghua University, Beijing, China, 100084. E-mail: {hmilyyz, ycdfwzy, horacehxw, zhaoyi.yuan31, jiahangok, i.haocao}@gmail.com.

Fan Dang is with the School of Software Engineering, Beijing Jiaotong University, Beijing. E-mail: dangfan@bjtu.edu.cn.

Yunhao Liu is with the Global Innovation Exchange, Tsinghua University, Beijing, China, 100084. E-mail: yunhaoliu@gmail.com.

Qiang Ma is with the QiYuan Lab, Beijing, China. E-mail: tsinghuamq@gmail.com.

*Zheng Yang is the corresponding author.

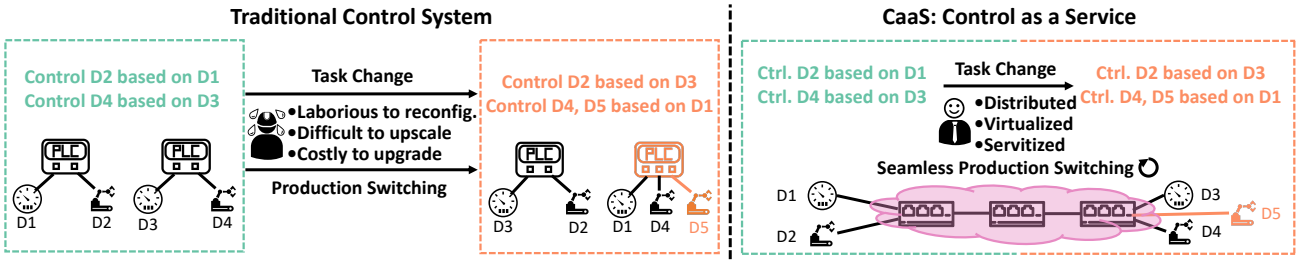


Figure 1: Traditional PLC-based control systems and Control-as-a-Service (CaaS) control systems. For traditional control system, when control tasks change, it is laborious, difficult, and costly to reconfigure, upscale, and upgrade current systems. In contrast, CaaS can realize seamless production switching by the distribution, virtualization, and servitization of control systems.

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. It decouples the tight binding among devices, controllers and tasks, 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 with a comprehensive mathematical formulation.
- We evaluate the performance of CaaS on testbeds based on both real-world networked control systems and simulated scheduling problems. 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. Furthermore, CaaS keeps a high scheduling success rate ($\geq$95%) across different topologies, outperforming the baseline by $\sim$30%.
- 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 future researches. (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.

Table I: Production line switching.

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

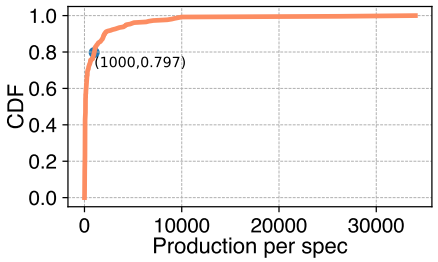


Figure 2: CDF of production per spec.

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. X.

This work does not raise any ethical issues.

II. Background and Motivation

A. Field Study

We have conducted the field study on the auto glass industry, at one of Fuyao Group's factories located in Fujian province, China. Fuyao is one of the world's largest automotive glass suppliers, with its products recognized and purchased by the world's top automobile manufacturers and major OEMs, such as Tesla, TOYOTA, BMW, Audi, *etc.* [10] With the diversification and personalization of customer requirements, the auto glass industry is in urgent need of flexible manufacturing. The auto glass manufacturing process can be divided into several general stages, which include raw glass pre-processing,

[1]Ziggo project: https://mobisense.github.io/ziggo_homepage/

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

printing, shaping, and quality control. However, the operations in each stage vary depending on the types and specifications of the auto glass, so do the control jobs related to these operations.

According to our investigation, 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 in the factory is depicted in Fig. 2. Over 79.7% are below 1,000 pieces. Each year, the investigated assembly line produces about 300 new types and specifications of vehicle glass. At present, the total number of glass specifications surpasses 10,000. 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, however, is challenging for existing control systems. PLCs are centralized in their operation, and devices must be directly connected to them. Additionally, the number of connected devices is limited by the IO ports on PLCs, also resulting in inflexibility of device connection.

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. This causes both extra downtime and costs.

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. Looking forward, flexible production expects the distribution, virtualization, and servitization of industrial control.

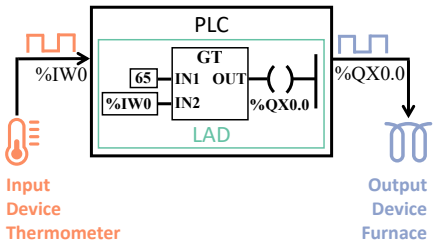


Figure 3: PLC and LAD.

B. PLC and LAD

Invented in the 1960s [11], PLC has grown to become the most widely used industrial control technology. A PLC is capable of controlling a variety of devices, including motors, lights, switchgear, *etc.* As a miniature computer, a PLC is capable of interacting with the physical world via its input/output (IO). A PLC executes control tasks periodically. Each execution cycle contains three stages: input scan, logic execution, and output writing. In the first stage, a PLC takes in inputs from sensor devices or human input points like buttons. In the second stage, it executes the internal logic programmed into it on its CPU and decides the outputs. In the last stage, it writes the outputs to output devices.

PLCs were originally developed to replace relay-based control systems. They are primarily used by electrical engineers. As a result, the primary programming interface of PLCs is graphical, resembling electrical diagrams. This programming language is referred to as Ladder Logic (LAD), and it is defined in IEC 61131-3 [12]. 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 and determine the output value. Finally, the output value is written out to control the furnace's on/off operation.

C. Industrial Networking and TSN

Industrial networking is responsible for deterministic communication between industrial controllers and devices. Initially, industrial networks rely on field bus technologies like Modbus [13], PROFIBUS [14], CAN [15]. Over the last decades, industrial Ethernet is gaining increasing acceptance. They are able to provide higher bandwidth and are based on standard Ethernet, making the industrial systems more interoperable. These technologies includes Beckhoff Automation's EtherCAT [16], Siemens' PROFINET [4], B&R's POWERLINK [5], and IEEE TSN [6]–[9]. They are designed for the existing PLC-based control systems, but can also be utilized in the CaaS architecture.

Among them, TSN is recognized as the most promising technology for the next generation industrial networks by major industrial technology companies, including Cisco, Marvell, and NXP [17]. Fig. 4 illustrates two major 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 [6]. Second, standard Ethernet switches transmit data as long as no higher-priority packets are waiting. IEEE 802.1Qbv [7] enables TSN switches to reserve dedicated time slots for critical

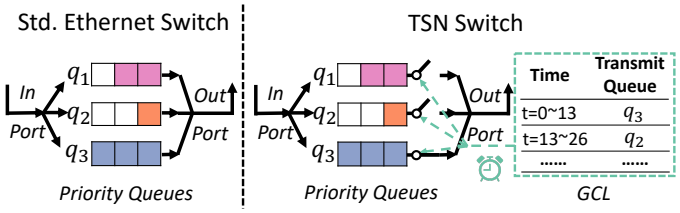


Figure 4: Standard Ethernet switch and TSN switch.

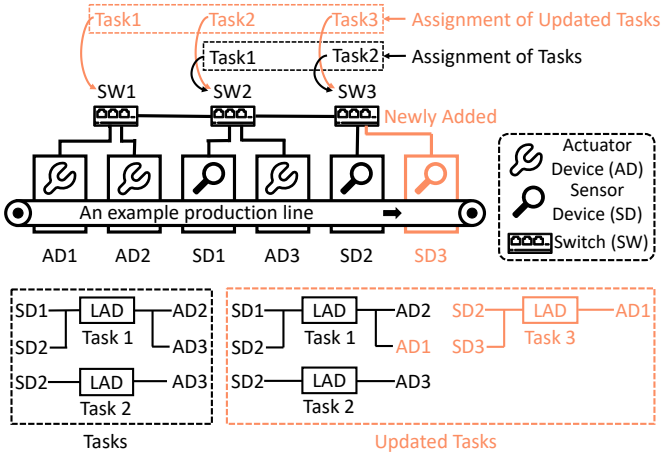


Figure 5: A working example of CaaS.

traffic, such as industrial control data. The reserved slots are specified by a schedule called the Gate Control List (GCL) contained within TSN switches. In the TSN network protocol stack, centralized Network Configuration (CNC), defined in IEEE 802.1Qcc [18], is responsible for network management. It generates a global schedule that specifies when data traffic should pass through each network link. Then CNC distributes the associated schedule, *i.e.*, gate control list (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 technique details. Then we introduce the architecture of CaaS in Sec. III-B.

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. Task 1 receives sensing data from SD1 and SD2, and then determines the outputs to control actuator devices AD2 and AD3 using the specified control logic. SD2 is used as the input device for Task 2 and AD3 is used as the output device. CaaS switches combine data transmission and task computation. CaaS CNC schedules (1) where (in which PLC) and when tasks are executed, and (2) the corresponding data transmission over network links. As illustrated in Fig. 5, 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. As shown in Fig. 5, the newly added SD3 can be connected to a nearby switch to join the network. Then CaaS CNC recalculates the schedule for the updated tasks and instantly deploys it to the whole network. Take a specific change as an example, SW2 and SW3 are both proper positions for the deployment of Task 2 (SD2→AD3). In updated tasks, Task 2 is moved from SW3 to SW2 since Task 3 is assigned to SW3. When the start time specified by CaaS CNC arrives, all switches in the network begin executing the new schedule.

B. System Architecture

We demonstrate the overall architecture of CaaS in Fig. 6. CaaS virtualizes the entire industrial control network into a universal controller. It allows upper-level control tasks and lower-level industrial devices easily move from conventional systems to the CaaS framework. Instead of binding to a specific PLC controller, control tasks can be scheduled and handled by any CaaS switches, facilitating flexible production line switching.

Inspired by the design philosophy of Software-Designed Networking (SDN), the CaaS architecture consists of four distinct layers: *application plane* with various industrial control tasks; *control plane* that schedules the network traffic and control tasks; *computation & forwarding plane* with networked CaaS switches; and *device plane* with industrial devices like sensors and mechanical arms.

In the *control plane*, a CaaS CNC runs the joint task and traffic scheduling algorithm, which will be presented in detail in Sec. V. It distributes the computed schedules to the network's switches and devices via NETCONF [19]. NETCONF is an IETF protocol for configuring network nodes. It can be used in IEEE 802.1Qcc to configure TSN switches and devices. We extend it to include not only traffic schedules but also task schedules.

In the *computation & forwarding plane*, the customized CaaS switches are connected as a control network. They execute the control tasks and forward the critical traffic deterministically based on the schedule provided by *control plane*. We sketch the design of the CaaS switch on the right of Fig. 6. It 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).

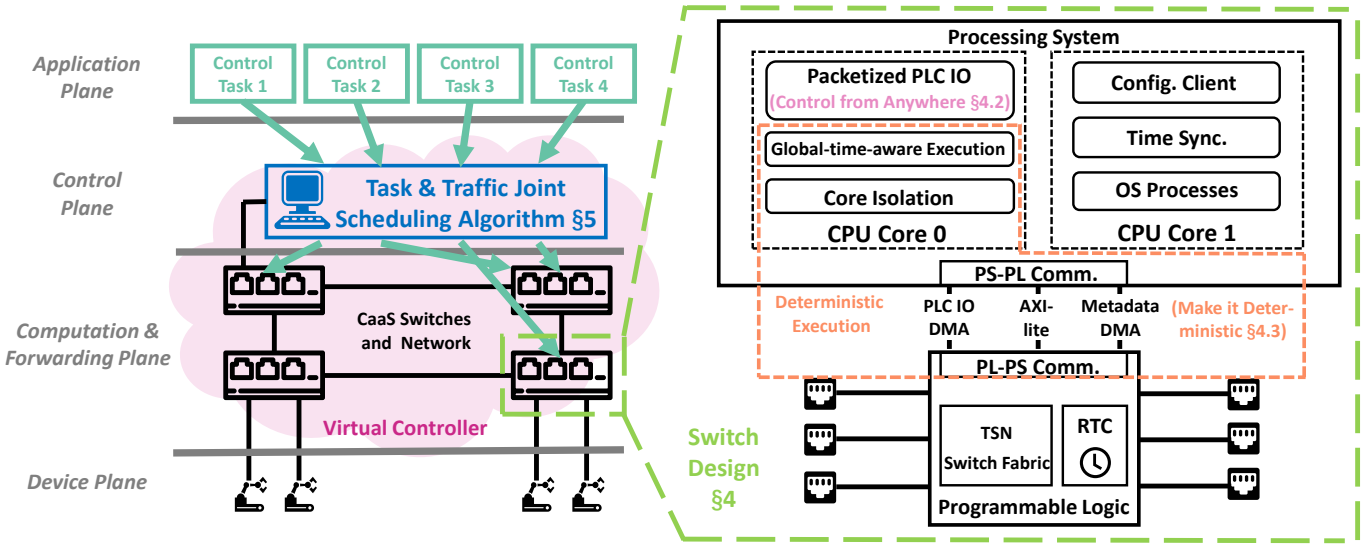


Figure 6: Overview of CaaS architecture.

PL implements functions that require high data rates or precise timing. Specifically, it implements the data transmission functions in accordance with IEEE TSN standards. Packets that are not destined for this switch get transmitted to output ports directly by PL. 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.

C. CaaS Workflow Example

When a new task, *e.g.* Task 3 in Fig. 5, needs to be added to the production line, the following process will take place:

- First, in the control plane, the Task & Traffic Joint Scheduling Algorithm is conducted on CaaS CNC to reallocate bandwidth and computational resources for all tasks, including the new one.
- Once a new global schedule is obtained, the production line stops, and all tasks in the network are paused. The new schedule, including the new task, is distributed to CaaS switches in the computation & forwarding plane.
- Finally, the production line is restarted. All tasks resume their operations and the new task is successfully added.

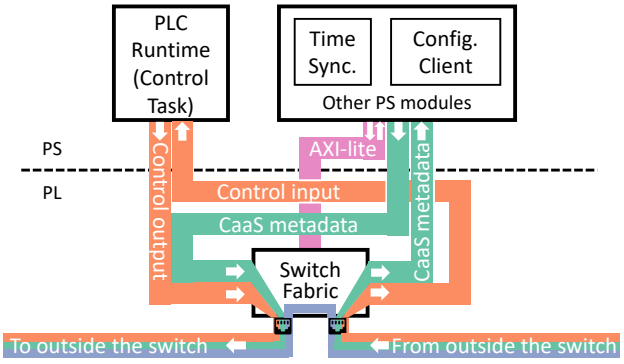


Figure 7: Workflow of CaaS switch. The relationship between three sources of incoming packets and three sources of outgoing packets is marked by different colors. The packets sourcing from "control output" and "CaaS metadata" are forwarded to "outside the switch". The packets from outside the switch are processed differently according to their types.

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): (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

This article has been accepted for publication in IEEE Journal on Selected Areas in Communications. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2025.3574598

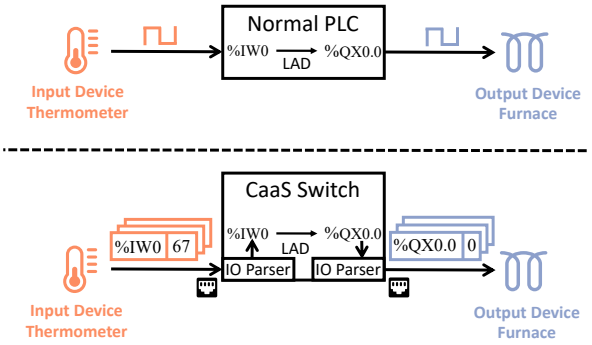


Figure 8: Wired PLC IO and Packetized PLC IO.

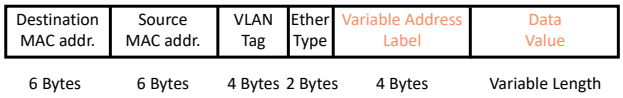


Figure 9: Frame format of Packetized PLC IO.

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. These modules may need to retrieve further related information from PL or change the values of some registers in PL. This is done through the AXI-lite interface between PS and PL.

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.

Before we dive into its details, we first introduce some preliminaries about how IO is represented in PLC and LAD. As shown in the upper part of Fig. 8, a thermometer is connected to the PLC as an input device. A furnace is connected as an output device. 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 [20] for the detailed naming conventions of variables). In every execution cycle, a PLC first reads the inputs and assigns the values to the bound variables, and then executes the LAD program. After the program finishes, the results are written out to output devices through the associated variables.

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. The frame format is shown in Fig. 9. 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.

After receiving the frame carrying input data, the *Packetized PLC IO* module binds the values to the LAD variables

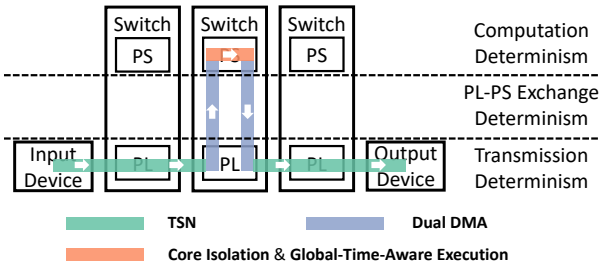


Figure 10: End-to-end determinism is guaranteed by TSN and three proposed designs.

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. 10, 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. *Core Isolation* and *GTA Execution* together ensure that the finish time of control tasks is also deterministic. 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. Thus, end-to-end determinism is achieved. Please note that CaaS represents a general industrial networking architecture. Although the current system implementation adopts TSN for transmission determinism, other industrial Ethernet protocols, such as PROFINET and EtherCAT, are also applicable within the CaaS framework.

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. Since these two DMA are physically independent, they will not interfere with each other.

Core Isolation: After the control input data arrives at PS, PLC runtime will start execution at the scheduled time. 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. *Dual DMA* decouples the transmission of CaaS metadata and control data, which makes it possible to further isolate the processing of these two kinds of data by *Core Isolation*.

GTA Execution: In CaaS, data transmission and task execution follow a global schedule configured by CaaS CNC. The PLC runtime we use is based on OpenPLC [21], which launches the execution according to CPU time, instead of the global synced time. To solve this problem, 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 design in the last section gives CaaS switches the ability to transmit data packets or execute control tasks at a specified time, *i.e.*, the ability to follow a predetermined schedule. In this section, we introduce the *Task & Traffic Joint Scheduling Algorithm* that calculates this global schedule.

In CaaS, input devices send data periodically to the controller switches, which then send the results to output devices. 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. Fig. 11 shows a schedule for an example CaaS system. There are two tasks. $Task1$ is scheduled on switch $SW2$ and $Task2$ is scheduled on $SW1$. As shown in the lower part of Fig. 11, the time when tasks are executed on switches and when data are transmitted on links are all specified in the schedule.

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.

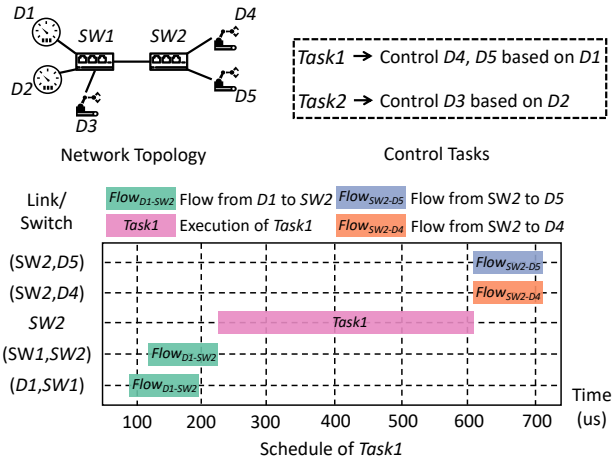


Figure 11: An example schedule of a CaaS System. $Task1$ controls $D4$ and $D5$ based on $D1$. The schedule of $Task1$ is shown in the lower part. $Task1$ is assigned to $SW2$. The input flow from $D1$ to $SW2$ passes link $(D1, SW1)$ and $(SW1, SW2)$. Its scheduled time slots on these links are marked in the figure. When the input data arrive at $SW2$, $Task1$ starts execution. After $Task1$ finishes, two time slots on link $(SW2, D4)$ and $(SW2, D5)$ are reserved to transmit two output flows.

B. Formulation

Similar to [22], the objective of this joint scheduling problem is to determine a scheduling table for the data transmission and computation of each task in J, such that, while satisfying constraints like task deadlines, non-overlapping resources, and precedence dependencies, the total delay of all tasks is minimized. The Satisfiability Modulo Theory (SMT) is the problem of determining whether a set of mathematical formulas is satisfiable. In this subsection, we model the joint scheduling problem as an SMT problem, by detailing the sources and formulations of each constraint. Finally, the joint scheduling problem can be solved with 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. For example, the transmission of input flows may be scheduled near the end of the period. As a result, the control tasks will be delayed to the next period. In this case, the value of $j_i.o$ should be 1, otherwise, it is 0. First, the tasks assigned to the same host cannot overlap with each other in time dimension:

$$
\begin{aligned}
&\forall j_a \in J, j_b \in J, a \neq b: \\
&P^s \leftarrow lcm(j_a.P, j_b.P) \\
&\forall u \in \{u \in \mathbb{N} | u \leq \frac{P^s}{j_a.P}\}, \forall v \in \{v \in \mathbb{N} | v \leq \frac{P^s}{j_b.P}\}: \\
&\quad AddConstraints(Imply(j_a.host = j_b.host, \\
&\quad j_a.start + u \times j_a.P + j_a.T < j_b.start + v \times j_b.P \vee \\
&\quad j_b.start + v \times j_b.P + j_b.T < j_a.start + u \times j_a.P))
\end{aligned}
\tag{1}
$$

where lcm calculates the least common multiple of arguments. $AddConstraints$ adds the constraints to the SMT model.

$Imply$ is a type of SMT constraint, which means that the first argument implies the second argument.

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 [23]:

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

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, similar to the meaning of $j_i.o$.

When a flow passes network links, the scheduled slots on successive links cannot precede those on former links:

$$\begin{aligned} &\forall j_i \in J, \forall f_{ik} \in F_i : \\ &\forall (s,e,c) \in \{(s,e,c) \in V \times V \times V | \\ &\qquad (s,e) \in E \wedge (e,c) \in E\} : \\ &AddConstraints(Imply(x_{ikse} = 1 \wedge x_{ikec} = 1, \\ &(t_{ikec} + o_{ikec} \times j_i.P) - (t_{ikse} + o_{ikse} \times j_i.P) \geq e.d) \end{aligned} \tag{2}$$

where $e.d$ is the forwarding delay of switch e. Additionally, the time slots reserved for different flows on the same link cannot overlap with each other. AF represents the set of all the flows and $f_{ik}.l$ represents the time needed to transmit the input or output data:

$$\begin{aligned} &\forall (f_{ik}, f_{wz}) \in \{(f_{ik}, f_{wz}) \in AF \times AF | i \neq w \vee k \neq z\} : \\ &P^s \leftarrow lcm(j_i.P, j_w.P) \\ &\forall u \in \{u \in \mathbb{N} | u \leq \frac{P^s}{j_i.P}\}, \forall v \in \{v \in \mathbb{N} | v \leq \frac{P^s}{j_w.P}\} : \\ &\forall (s,e) \in E : \\ &AddConstraints(Imply(x_{ikse} = 1 \wedge x_{wzse} = 1, \\ &(t_{wzse} + v \times j_w.P) - (t_{ikse} + u \times j_i.P) \geq f_{ik}.l \vee \\ &(t_{ikse} + u \times j_i.P) - (t_{wzse} + v \times j_w.P) \geq f_{wz}.l)) \end{aligned} \tag{3}$$

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:

$$\begin{aligned} &\forall j_i \in J, \forall f_{ik} \in F_i, \forall h \in V_{sw} : \\ &m \leftarrow \begin{cases} f_{ik}.src, & \text{if } f_{ik} \text{ is an input flow}, \\ f_{ik}.dst, & \text{if } f_{ik} \text{ is an output flow}, \end{cases} \\ &sp \leftarrow shortestpath(h, m). \\ &\forall (s,e) \in E, \\ &AddConstraints(Imply(j_i.host = h \wedge (s,e) \in sp, x_{ikse} = 1)), \\ &AddConstraints(Imply(j_i.host = h \wedge (s,e) \notin sp, x_{ikse} = 0)), \end{aligned} \tag{4}$$

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:

$$\begin{aligned} &\forall j_i \in J, \forall f_{ik} \in F_i, \forall (s,e) \in E : \\ &AddConstraints(Imply(x_{ikse} = 0, t_{ikse} = 0 \wedge o_{ikse} = 0)). \end{aligned} \tag{5}$$

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:

$$\begin{aligned} &\forall j_i \in J, \forall f_{ik} \in F_i^{src} : \\ &endt \leftarrow \sum_{(s,e) \in E} If(x_{ikse} = 1 \wedge j_i.host = e, \\ &\qquad t_{ikse} + o_{ikse} \times j_i.P + f_{ik}.l, 0), \\ &AddConstraints(j_i.start + j_i.o \times j_i.P \geq endt), \end{aligned} \tag{6}$$

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}. In Eq.(6), we use a combination of $\sum$ and If to represent the arrival time. It considers all the possible last links for the input flows. Only the actual scheduled link will contribute to the sum.

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:

$$\begin{aligned} &\forall j_i \in J, \forall f_{ik} \in F_i^{dst} : \\ &startt \leftarrow \sum_{(s,e) \in E} If(x_{ikse} = 1 \wedge j_i.host = s, \\ &\qquad t_{ikse} + o_{ikse} \times j_i.P, 0), \\ &AddConstraints(j_i.start + j_i.o \times j_i.P + j_i.T \leq startt). \end{aligned} \tag{7}$$

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:

$$\begin{aligned}
&\forall j_i \in J: \\
&srct_i \leftarrow min(\{\sum_{(s,e)\in E, s=f_{ik}.src} (t_{ikse} + o_{ikse} \times j_i.P) \\
&\qquad |f_{ik} \in F_i^{src}\}), \\
&dstt_i \leftarrow max(\{\sum_{(s,e)\in E, e=f_{ik}.dst} (t_{ikse} + o_{ikse} \times j_i.P) \\
&\qquad + f_{ik}.l | f_{ik} \in F_i^{dst}\}), \\
&AddConstraints(dstt_i - srct_i \leq j_i.MD),
\end{aligned} \tag{8}$$

The scheduling variables representing the flows' departure and arrival time depend on the scheduling of tasks. Eq.(8) sums up over all possible links and Eq.(5) guarantees that only the scheduled links contribute to the $\sum$ in Eq.(8).

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

$$Minimize(\sum_{j_i \in J} dstt_i - srct_i), \tag{9}$$

where $dstt$ and $srct$ are defined in Eq.(8).

The SMT problem defined above is implemented using the Python API of the Z3 SMT solver (Version 4.8.13) [24].

VI. Implementation

A. CaaS Switch

We implement CaaS switches on Xilinx ZYNQ-7000 SoC [25]. 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 [7], [18], [26], [27]. 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. GCL is implemented using BRAM in PL and it stores the schedule issued from CNC. A transmission selector is also implemented to execute this schedule and control the state of each frame queue.

Time Synchronization: We implement the time synchronization protocol defined in IEEE 802.1AS [6]. The implementation adopts a PS-PL cooperation design. 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 [21]. 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*. *Core Isolation* is enabled by building the Linux OS for CaaS with boot parameter `isolcpus=<cpu number>`.

B. CaaS Evaluation Device:

We design and implement the CaaS evaluation device to emulate real-world sensor and actuator devices. It can measure every I/O packet's end-to-end latency with sub-100ns accuracy. Similar to the CaaS switch, the CaaS device is implemented on Xilinx ZYNQ-7000 SoC and complies with IEEE 802.1AS time synchronization protocol. In addition, we implement the following components:

On-Time Packet Generation: To emulate sensor devices' data sending behavior, we implement an on-time packet generation module in PL. The module sends packets periodically with specified destinations and contents, following the global schedule provided by CaaS CNC.

Packet-Level Latency Analysis: To record packets' end-to-end latency at hardware-level ($8ns$) timing accuracy, we implement a timestamping module between physical Ethernet ports and all the other hardware logic in PL. We also extend the frame format in Sec. IV-B with two 64-bit timestamp fields at the tail. Every time an I/O packet arrives at or leaves a CaaS device, the timestamping modules will record the current time in one of the two timestamp fields. After timestamping, the received I/O packets are forward to PS directly through DMA for further analysis.

VII. Evaluation

A. Overall System Performance

In this section, we evaluate the overall system performance of CaaS on three testbeds, which are built according to real-world control systems.

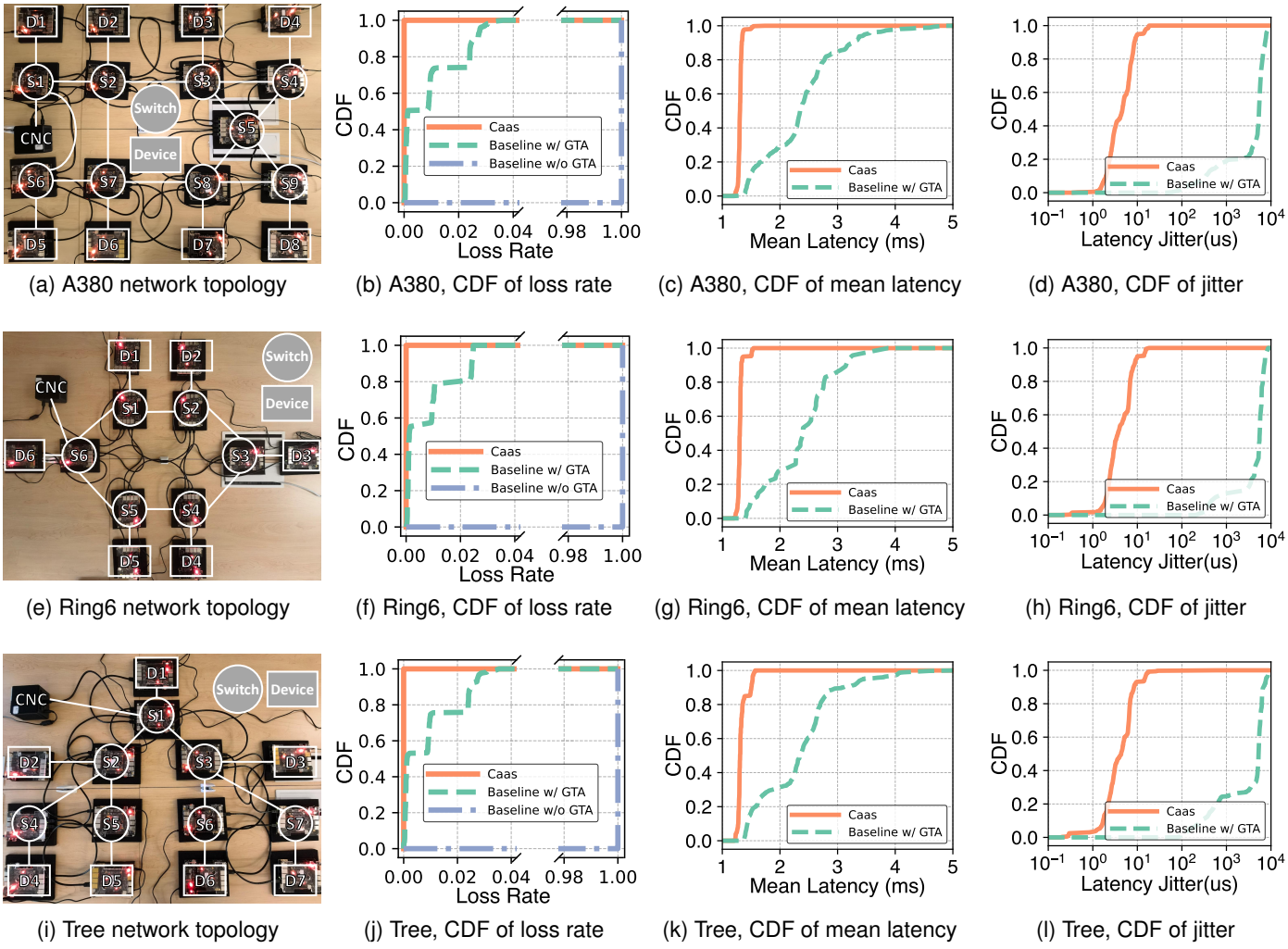
1) Setup: The setup of testbeds are as follows:

Topology: The first testbed A380 uses the topology shown in Fig. 12a, which is a simplified version of the control network used on Airbus A380 [28]. It consists of 9 switches and 8 devices. The second testbed Ring6 uses the ring topology shown in Fig. 12e, consisting of 6 switches and 6 devices. The third testbed, *i.e.*, Tree, is shown in Fig. 12i, consisting of 7 switches and 7 devices. Ring and tree topologies are also common in industrial control networks [29], [30].

Baseline: We compare CaaS with two baselines. 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 [31], [32]. We randomly generate n tasks with m sensor devices and q actuator devices, where n equals the number of CaaS switches, m and q are randomly chosen from 1 to 4 [28]. 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.



(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

(i) Tree network topology (j) Tree, CDF of loss rate (k) Tree, CDF of mean latency (l) Tree, CDF of jitter

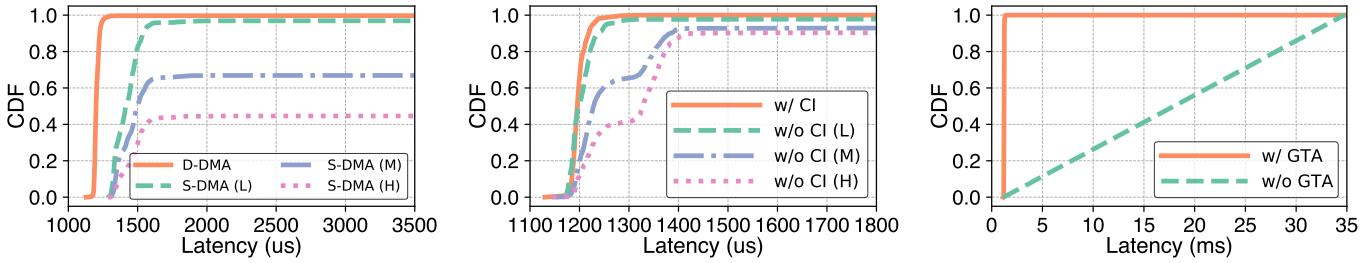
Figure 12: Overall performance on different testbeds.

2) Results: Fig. 12 show results on different testbeds. Overall, CaaS achieves absolute packet delivery, 42-45% lower latency, and three orders of magnitude lower jitter on all three testbeds.

Fig. 12b, 12f, and 12j display the loss rate results. On all three 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 all three testbeds, while for *Baseline w/ GTA*, the 99 percentile loss rate is 3.18% (A380), 2.47% (Ring6), and 3.21% (Tree). This indicates that the proposed designs in Sec. IV-C can substantially improve tasks' end-to-end determinism.

The distribution of mean latency is shown in Fig. 12c, 12g, and 12k. It is clear that CaaS shows more concentrated latency distribution and lower average latency than baseline. Taking Fig. 12c 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 on three testbeds. 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. In addition, we observe that there is a sudden change of latency at upper-left corner of Fig. 12c, 12g, and especially 12k. This is because some stochastic tasks tend to have longer paths from source to destination, and the Tree topology is more likely to have such tasks. To sum up, CaaS outperforms the baseline in both real time and determinism.

As shown in Fig. 12d, 12h, and 12l, the jitter of CaaS is almost negligible compared to the baseline. Specifically, the median jitter on A380, Ring6, and Tree is $4.6\mu s$, $3.7\mu s$, $4.4\mu s$ for CaaS, and $5.4ms$, $5.6ms$, $5.2ms$ for baseline, respectively. The jitter of CaaS is over three orders of magnitude lower than Baseline. In addition, 75.4%-87% of the tasks have jitter greater than $1ms$ in the results of baseline, which is already unacceptable for industrial applications. The main reason is the absence of *Core Isolation* in the baseline, which causes uncertain execution time. Thus, lots of tasks cannot be finished before scheduled transmission time slot, even though the PLC runtime gets input packets on time.



(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 13: 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.

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. 13. 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. 13a. We also study the impacts of traffic load on the two methods. The results show that 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. 13a 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. Additionally, due to the severe queuing of single DMA, the transmission latency (without computation latency) of *D-DMA* is 51.3% lower than that of *S-DMA* even under the lightest traffic load.

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. 13b. 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. As shown in Fig. 13b, the loss rate of *w/o CI* rises from 2.1% to 9.7% with increased CPU usage.

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. 13c. 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.

C. Scheduling Algorithm Performance

1) Setup: We mainly investigate the schedulability of our proposed scheduling algorithm (hereinafter called *CaaS*) and the two-step scheduling algorithm (hereinafter called *Baseline*) described in Sec. VII-A1. We test these two algorithms on the network topologies of our three testbeds: A380, Ring6, and Tree. On each topology, we produce 100 groups of stochastic tasks following the rules described in Sec. VII-A1. The number of tasks in each group varies in $\{5, 10, 15, 20\}$. We evaluate the schedulability by the proportion of groups that have a feasible schedule under each method.

2) Results: Fig. 14 illustrates the superiority of CaaS over Baseline. The figures show the scheduling success rates of CaaS and Baseline on different topologies. Clearly, CaaS keeps a high success rate (no less than 95%) as the number of tasks increases, whereas Baseline's success rate drops significantly. Specifically, CaaS improves the schedulability by at most 29%, 38%, and 55% on A380, Ring6, and Tree, respectively. The reason lies in the fact that CaaS considers the scheduling of tasks and traffic at the same time, instead of fixing the task assignment first. Thus, CaaS can find feasible schedules more likely than Baseline. In addition, we find that Baseline's success rates on A380 topology are much higher than those on Ring6 or Tree topology. This results from the complexity of A380 topology: there are multiple paths between each pair of nodes in A380 topology, which helps improve the chances of finding a feasible schedule.

This article has been accepted for publication in IEEE Journal on Selected Areas in Communications. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2025.3574598

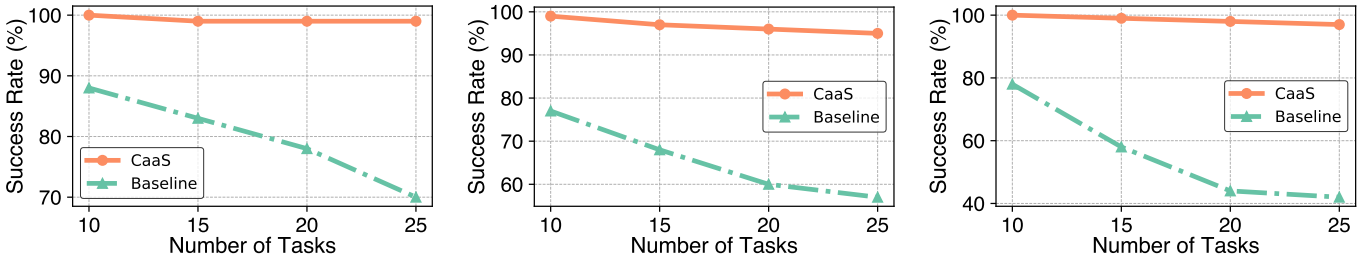


Figure 14: Scheduling algorithm success rate.

VIII. Related Work

Virtual PLC: In recent years, some work tries to replace the hardware PLCs in existing industries with virtual ones running in virtual machines or containers. 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.* [33] provide a case study to build virtual PLCs in cloud servers in order to bring more agility to industrial automation systems. Hegazy *et al.* [34] 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 [33]. 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 [22], based on their previous work on other deterministic networks [35]–[37]. Besides, some work uses Integer Linear Programs (ILP) to formulate and solve the problem in different ways [38]–[40]. 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 [41]–[43], incremental backtracking [44], greedy randomized search [45], [46], or deep reinforcement learning [47] to increase the number of flows that can be scheduled under a fixed time budget. There are also ongoing efforts to extend TSN traffic scheduling to broader areas, such as wireless TSN [48] and TCP over TSN [49]. In a nutshell, prior work only focuses on traffic scheduling, while CaaS jointly considers task and traffic scheduling problems.

In-network Computation: The programmable data plane technologies like P4 [50] and its back-ends have started a new trend of in-network computation. They are widely adopted in network telemetry [51], traffic management [52], and packet scheduling or routing [53]. Besides, some work performs more general tasks like massive-scale data processing [54], and machine learning [55], [56] with programmable data planes. They mainly focus on throughput or latency improvement for network-bound applications in data centers. In contrast, CaaS is designed for deterministic industrial control. Some prior work also investigates offloading simple industrial control logic to programmable switches [57]–[59]. However, they only apply to specific scenarios, whereas CaaS provides a more general framework to support distributed, virtualized, and servitized industrial control. The capacity of existing programmable network devices is also insufficient even for simple coordinate transformation tasks [60].

IX. Discussion

We discuss the future work in this section.

Scheduling in the large-scale network. In addition to success rate, efficiency is another important metric for network traffic scheduling problems. In this work, we focus on scheduling success rates. By modeling the scheduling problem with SMT, we can ensure that all feasible solutions are explored, but this approach has exponential computational complexity. In our experiments, when the number of flows reached 100 or more, running the scheduling algorithm often took several hours or even days. So, it is the future work to design an efficient algorithm to improve the efficiency of the joint scheduling problem. In addition, popular deep learning techniques also have the potential for application in traffic scheduling algorithms, representing a direction for future research.

Intelligent industrial control. As mentioned in Sec. I , control tasks are evolving from simple relay logic to complex machine learning models. While transitioning CaaS architecture from traditional industrial control computing to deep-learning-based control computing only seems to demand some engineering efforts, there are many challenges that may arise in practice. It is nontrivial to ensure the real-time performance of model inference and to guarantee the deterministic transmission of model input data, such as images and point clouds. In other words, CaaS equipped with intelligent industrial control remains an important research issue for the future.

X. Conclusion

In this work, we propose CaaS, a new architecture for industrial control systems. CaaS transfers and distributes control tasks from dedicated controllers into network switches. In the design of CaaS, control is a service that can be called 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. We have provided public access to our code at https://doi.org/10.5281/zenodo.7489411.

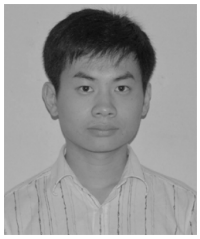
ACKNOWLEDGMENT

This work is supported in part by the National Key Research Plan under grant No.2021YFB2900100, the NSFC under grant No. 62372265, No. 62302259, and No. 62302254.

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] *Industrial networks - Profiles - Part 2-21: Additional real-time fieldbus profiles based on ISO/IEC/IEEE 8802-3 - CPF 21*, IEC Std. 61 784-2-21:2023, 2023.
[5] "Ieee standard for industrial hard real-time communication," *IEEE Std 61158-2017 (Adoption of EPSG DS 301)*, pp. 1–395, 2017.
[6] *Timing and Synchronization for Time-Sensitive Applications*, IEEE Std. 802.1AS, 2020.
[7] *Enhancements for Scheduled Traffic*, IEEE Std. 802.1Qbv, 2015.
[8] *Frame Preemption*, IEEE Std. 802.1Qbu, 2016.
[9] *Path Control and Reservation*, IEEE Std. 802.1Qca, 2015.
[10] F. Group. (2023) Company profile. [Online]. Available: https://www.fuyaogroup.com/en/about.html
[11] 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
[12] *Programmable controllers - Part 3: Programming languages*, IEC Std. 61 131-3:2013, 2013.
[13] T. M. Organization. (2012) Modbus application protocol. [Online]. Available: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf
[14] *Industrial networks - Profiles - Part 1-22: Fieldbus profiles - Communication Profile Family 22*, IEC Std. 61 784-1-22:2023, 2023.
[15] *Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical coding sublayer*, ISO Std. 11 898-1:2024, 2024.
[16] *Industrial communication networks - Fieldbus specifications - Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*, IEC Std. 61 158-1:2023, 2023.
[17] M. Research. (2021) Time-sensitive networking market. [Online]. Available: https://www.marketsandmarkets.com/Market-Reports/time-sensitive-networking-market-215000493.html
[18] *Stream Reservation Protocols (SRP) Enhancements and Performance Improvements*, IEEE Std. 802.1Qcc, 2018.
[19] R. Enns, M. Björklund, A. Bierman, and J. Schönwälder, "Network Configuration Protocol (NETCONF)," RFC 6241, Jun. 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6241
[20] OpenPLC. (2021). [Online]. Available: https://www.openplcproject.com/reference/plc-addressing/
[21] 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.
[22] 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.
[23] 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
[24] Z3Prover. (2021) Github repository of z3prover. [Online]. Available: https://github.com/Z3Prover/z3
[25] Xilinx. (2021) Socs with hardware and software programmability. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html
[26] *Bridges and Bridged Networks*, IEEE Std. 802.1Q, 2018.
[27] *Forwarding and Queuing Enhancements for Time-Sensitive Streams*, IEEE Std. 802.1Qav, 2009.
[28] 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.
[29] P. N. America. (2022) Industrial topology options and profinet. [Online]. Available: https://us.profinet.com/wp-content/uploads/2019/08/Topology.pdf
[30] W. Voss. (2019) Industrial ethernet guide - network topologies. [Online]. Available: https://copperhilltech.com/blog/industrial-ethernet-guide-network-topologies/
[31] PLC-City. (2021) Siemens simatic s7-1500. [Online]. Available: https://www.plc-city.com/shop/en/siemens-simatic-s7-1500.html
[32] 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
[33] 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.
[34] 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.
[35] 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.
[36] 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.
[37] 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.
[38] 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.
[39] 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.
[40] 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.
[41] F. Glover and M. Laguna, "Tabu search," in *Handbook of combinatorial optimization*. Springer, 1998, pp. 2093–2229.
[42] 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
[43] 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.
[44] 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.
[45] M. G. Resende and C. C. Ribeiro, "Grasp: Greedy randomized adaptive search procedures," in *Search methodologies*. Springer, 2014, pp. 287–312.
[46] 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.
[47] 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.
[48] D. Akhmetov, D. Das, D. Cavalcanti, J. Ramirez-Perez, and L. Cariou, "Scheduled time-sensitive transmission opportunities over wi-fi," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, 2022, pp. 1807–1812.
[49] R. Candell, K. Montgomery, M. Kashef Hany, S. Sudhakaran, and D. Cavalcanti, "Scheduling for time-critical applications utilizing tcp in

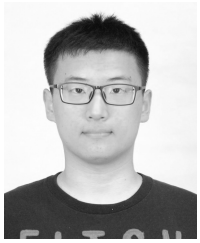
software-based 802.1qbv wireless tsn," in *2023 IEEE 19th International Conference on Factory Communication Systems (WFCS)*, 2023, pp. 1–8.
[50] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, Jul. 2014.
[51] Y. Zhang, Z. Liu, R. Wang, T. Yang, J. Li, R. Miao, P. Liu, R. Zhang, and J. Jiang, "CocoSketch: High-performance sketch-based measurement over arbitrary partial key query," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. Virtual Event USA: ACM, Aug. 2021, pp. 207–222.
[52] D. Kim, Z. Liu, Y. Zhu, C. Kim, J. Lee, V. Sekar, and S. Seshan, "TEA: Enabling State-Intensive Network Functions on Programmable Switches," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*. Virtual Event USA: ACM, Jul. 2020, pp. 90–106.
[53] V. Shrivastav, "Fast, scalable, and programmable packet scheduler in hardware," in *Proceedings of the ACM Special Interest Group on Data Communication*. Beijing China: ACM, Aug. 2019, pp. 367–379.
[54] T. Kohler, R. Mayer, F. Dürr, M. Maaß, S. Bhowmik, and K. Rothermel, "P4CEP: Towards In-Network Complex Event Processing," in *Proceedings of the 2018 Morning Workshop on In-Network Computing*, ser. NetCompute '18. New York, NY, USA: Association for Computing Machinery, Aug. 2018, pp. 33–38.
[55] G. Siracusano and R. Bifulco, "In-network Neural Networks," *arXiv:1801.05731 [cs]*, Jan. 2018.
[56] C. Zheng and N. Zilberman, "Planter: Seeding trees within switches," in *Proceedings of the SIGCOMM '21 Poster and Demo Sessions*. New York, NY, USA: Association for Computing Machinery, Aug. 2021, pp. 12–14.
[57] F. Geyer and M. Winkel, "Towards Embedded Packet Processing Devices for Rapid Prototyping of Avionic Applications," in *9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*, Toulouse, France, 2018.
[58] F. E. R. Cesen, L. Csikor, C. Recalde, C. E. Rothenberg, and G. Pongrácz, "Towards Low Latency Industrial Robot Control in Programmable Data Planes," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, Jun. 2020, pp. 165–169.
[59] A. Rahman, J. Jin, A. L. Cricenti, A. Rahman, and A. Kulkarni, "Communication-aware cloud robotic task offloading with on-demand mobility for smart factory maintenance," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 5, pp. 2500–2511, 2019.
[60] I. Kunze, R. Glebke, J. Scheiper, M. Bodenbenner, R. H. Schmitt, and K. Wehrle, "Investigating the Applicability of In-Network Computing to Industrial Scenarios," in *2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*. Victoria, BC, Canada: IEEE, May 2021, pp. 334–340.



Zheng Yang is an associate professor at Tsinghua University. His main research interests include Internet of Things and industrial Internet. He is a fellow of IEEE and a member of ACM.

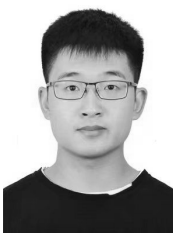
Zeyu Wang received his B.E. degree in School of Software from Tsinghua University in 2020. He is now a Ph.D. candidate in School of Software, Tsinghua University. His research interests include Time-Sensitive Networking and mobile computing

Xiaowu He received his B.E. degree in Schoole of Computer Science and Engineering from University of Electronic Science and Technology of China in 2019, and his Ph.D. degree in School of Software, Tsinghua University in 2024. His research interests include Time-Sensitive Networking, edge computing, and Internet of Things.



Yi Zhao Yi Zhao received his B.E. and Ph.D. degree in School of Software from Tsinghua University in 2017 and 2022 respectively. His research interests include Internet of Things, Edge Computing and TSN.

Fan Dang received his B.E. and Ph.D. degrees in software engineering from Tsinghua University, Beijing, in 2013 and 2018, respectively. He is an assistant professor in the School of Software Engineering, Beijing Jiaotong University, Beijing. He is a senior member of IEEE and a member of ACM. His research interests include industrial intelligence and edge computing.



Jiahang Wu received his B.E. degree from the School of Software, Beijing Jiaotong University, in 2020. He received his M.A. degree in School of Software, Tsinghua University in 2023. His research interests include the Internet of Things and Time-Sensitive Networking.

Hao Cao received his B.E. degree in College of Intelligence and Computing from Tianjin University in 2019. He received his Ph.D. degree in School of Software, Tsinghua University in 2024. His research interests include Internet of Things and mobile computing.

Yunhao Liu received his B.S. degree in Automation Department from Tsinghua University, and an M.A. degree in Beijing Foreign Studies University, China. He received an M.S. and a Ph.D. degree in Computer Science and Engineering at Michigan State University, USA. Yunhao is now the Dean of the Global Innovation Exchange Institute, Tsinghua University.

Qiang Ma received the B.S. degree from the Department of Computer Science and Technology, Tsinghua University, China, in 2009, and the Ph.D. degree from the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, in 2013. He is now an assistant researcher in QiYuan Lab. His research interests include wireless sensor networks, mobile computing and privacy.