IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 30, 2021, accepted March 21, 2021, date of publication March 26, 2021, date of current version April 7, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3069142

Communication Scheduling for Control Performance in TSN-Based Fog Computing Platforms

MOHAMMADREZA BARZEGARAN, (Graduate Student Member, IEEE), AND PAUL POP, (Member, IEEE)
Department of Applied Mathematics and Computer Science, Technical University of Denmark, 2800 Kongens Lyngby, Denmark

Corresponding author: Mohammadreza Barzegaran (mohba@dtu.dk)

This work was supported by the European Union's Horizon 2020 Research and Innovation Programme through the Marie Skłodowska-Curie, FORA—Fog Computing for Robotics and Industrial Automation, under Grant 764785.

ABSTRACT In this paper we are interested in real-time control applications that are implemented using Fog Computing Platforms consisting of interconnected heterogeneous Fog Nodes (FNs). Similar to previous research and ongoing standardization efforts, we assume that the communication between FNs is achieved via the IEEE 802.1 Time Sensitive Networking (TSN) standard. We model the control applications as a set of real-time flows, and we assume that the messages are transmitted using scheduled traffic that is using the Gate Control Lists (GCLs) in TSN. Given a network topology and a set of control applications, we are interested to synthesize the GCLs for messages such that the Quality-of-Control (QoC) of control applications is maximized and the deadlines of real-time messages are satisfied. We have proposed a Constraint Programming (CP)-based solution to this problem, and developed an accurate analytical model for QoC, which, together with a metaheuristic search employed in the CP solver can drive the search quickly towards good quality solutions. We have evaluated the proposed strategy on several test cases including realistic test cases and also validate the resulted GCLs on a TSN hardware platform and via simulations in OMNET++.

INDEX TERMS Fog computing, optimization, quality-of-control, TSN.

I. INTRODUCTION

We are at the beginning of a new industrial revolution (Industry 4.0), which will bring increased productivity and flexibility, mass customization, reduced time-to-market, improved product quality, innovations and new business models. However, Industry 4.0 will only become a reality through the convergence of Operational and Information Technologies (OT & IT), which are currently separated in a hierarchical pyramid (Purdue Reference Model [1]) and use different computation and communication technologies. OT consists of cyber-physical systems that monitor and control physical processes that manage, e.g., automated manufacturing, critical infrastructures, smart buildings and smart cities. These application areas are typically safety-critical and real-time, requiring guaranteed non-functional properties, such as, real-time behavior, reliability, availability, safety, and security and often required to show compliance to industry specific standards. OT uses proprietary solutions, imposing severe restrictions on the information flow.

Instead, a new paradigm, called Fog Computing, is envisioned as an architectural means to realize the IT/OT convergence in Industrial IoT [2], which cannot be realized using Cloud Computing. According to NIST, ''Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources [...] that can be rapidly provisioned and released with minimal management effort or service provider interaction'' [3]. The OpenFog IEEE standard defines *Fog Computing* as a ''system-level architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Things'' [4]. We define *Edge Computing* is as a new architectural paradigm in which the resources of an edge server are placed at the edge

The associate editor coordinating the review of this manuscript and approving it for publication was Songwen Pei.

of the Internet, in close proximity to cyber-physical systems, mobile devices, sensors and IoT endpoints.

With Fog Computing, communication devices, such as switches and routers are extended with computational and storage resources to enable a variety of communication and computation options. Fog Computing will enable a powerful convergence, unification and standardization at the networking, security, data, computing, and control levels. It will lead to improved interoperability, security, more efficient and rich control, and higher manufacturing efficiency and flexibility [5]. The vision is to virtualize the control (implemented as software tasks exchanging messages) and achieve the same level dependability, i.e., non-functional properties such as reliability, timeliness and security as the one taken for granted in *OT*, achieved via dedicated hardware and software solutions.

The convergence of IT and OT will be supported by: the increased usage of IP-protocols, e.g., standardized Deterministic Ethernet solutions from IEEE Time Sensitive Networking (TSN) Task Group [6], upcoming 5G wireless standards [7], and interoperability standards such as OPC Unified Architecture (OPC UA) [8], all integrated into a Fog Computing Platform (FCP), see Fig. 1, which brings computation, communication and storage closer to the edge of the network. Several initiatives are currently working towards realizing this vision [9], [10].

The integration of computational and storage resources into the communication devices is realized in the Fog Node (FN), see Fig. 1. In many applications, including industrial automation and robotics, several layers of FNs with differing computation, communication and storage capabilities will evolve, from powerful high-end FNs to low-end FNs with limited resources. Researchers have started to propose solutions for the implementation of FNs [5], [9], [11] and FN solutions have started to be developed by companies [9], [10], [12].

Regarding the communication infrastructure, today, industry uses mostly proprietary protocols [13] that lock customers into the product portfolio of individual product vendors, impairing interoperability. However, industry is moving towards using standardized solutions to connect the FNs to each other and to the machines [14], i.e., IEEE 802.1 TSN [6], see Fig. 1. Such an FCP allows to increase the spatial distance between the physical process and the FN that controls it, allowing the control functions to be executed remotely on the FN. However, the way the FCP and, in particular, the TSN communication infrastructure is configured has an impact on the control performance of the control applications. In our case, we consider high-end Fog Nodes connected to industrial systems and placed at the edge of the network, similar to edge servers, interconnected via TSN.

TSN consists of a set of amendments to the IEEE 802.1 Ethernet standard to provide features useful for real-time and safety critical applications.[1] An FCP hosts applications

[1]The references for all sub-standards can be easily found via IEEE Xplore

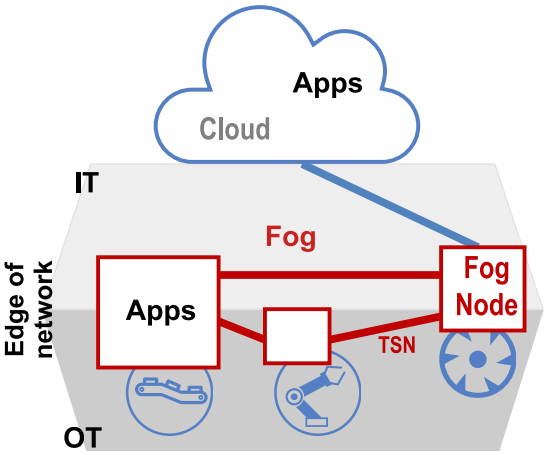

FIGURE 1. Fog Computing Platform: Boxes represent fog nodes, placed at the edge of network where OT and IT converge, connected to each other and to the Cloud in IT and to the industrial "thing" in OT, and running applications (Apps). Thick lines are physical TSN links.

of mixed-criticalties, which have different requirements, in terms of safety, timeliness and control performance. TSN supports multiple traffic types, and hence, is suitable for mixed-criticality applications running on an FCP. Applications with tight timing constraints typically use Scheduled Traffic (ST) implemented via IEEE 802.1Qbv, which defines a Time-Aware Shaper (TAS) mechanism that enables the scheduling of messages based on a global schedule table. The scheduling relies on a clock synchronization mechanism 802.1ASrev [15], which defines a global notion of time. Thus the devices are synchronized, and the global schedule is formed. Applications that need bounded latency but do not have stringent latency and jitter requirements can use the IEEE 802.1BA Audio Video Bridging Systems (AVB) traffic type. Best-Effort (BE) traffic compliant with IEEE 802.3 Ethernet can be used for non-critical applications that do not need timing guarantees. ST traffic has the highest priority, followed by AVB and BE. AVB mechanisms are intended to prevent the starvation of lower priority BE flows.

In this paper we address control applications virtualized on a distributed FCP, which are implemented as tasks running on FNs that exchange messages over TSN. We assume, similar to the related work, that the messages use the ST traffic type. In this context, the scheduling of ST messages has a strong impact on the Quality-of-Control (QoC), i.e., the control performance [16]. Given the network topology of the FCP, the set of mixed-criticality applications, for which we know their communication flows and their routing, we are interested to synthesize the TSN ST communication schedules such that the QoC is maximized and the mixed-criticality application requirements, e.g., deadlines, are satisfied. We have proposed a Constraint Programming (CP)-based solution for deriving the ST communication schedules. We have addressed the problem of scheduling of tasks on an FCP for QoC [17], which is orthogonal to the message scheduling problem. However, to facilitate the

integration of tasks and message schedules, our CP implementation also aims at supporting the integration of tasks and messages by creating space in the communication schedule timelines, where tasks need to execute.

A. CONTRIBUTIONS

The related work, discussed in Sect. VII, has shown that the communication synthesis has a strong impact on control performance. TSN has become a de-facto standard in several areas, including industrial applications. Although there has been much work in scheduling ST traffic in TSN, very few researchers have addressed scheduling in TSN for control performance [16], [18]. Compared to these works, the main contributions of this paper are as follows. We formulate the ST scheduling for QoC as an optimization problem, and propose a scalable CP-based solution to solve it. Our CP formulation considers all the relevant constraints of TSN, e.g., frame isolation, forwarding delay, resulting in realistic schedules that have been validated via simulations in *OMNET++* and on a TSN hardware platform. We consider a more realistic model of control applications and provide more accurate measure of QoC compared to previous work, based on JitterTime. JitterTime uses time consuming *simulations* of the control application behavior, and hence they cannot be integrated into a CP solver since the search will not scale. Thus, we proposed a novel *analytical model* for the QoC evaluation within the CP formulation. In addition, we have used a metaheuristic search strategy in the CP-solver to quickly obtain good quality solutions, enabling us to handle large test cases.

B. OUTLINE OF THE PAPER

The system model is presented in Sect. II where architecture, application and the internals of a TSN switch are described. We formulate our problem in Sect. III. An introduction to control theory is presented in Sect. IV. In Sect. V, the details of our proposed method are given. We evaluate our proposed method in Sect. VI on several test cases. The related work in presented in Sect. VII and Sect. VIII concludes the paper.

II. SYSTEM MODEL

This section presents the architecture and application models. Table 1 summarizes the notation used. The application model consists of a set of periodic messages that are sent via *flows* over a distributed Fog-based architecture that consists of *end systems* interconnected via *links* and *switches* that use TSN.

A. ARCHITECTURE MODEL

The architecture is modeled as a directed graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V} = \mathbf{ES} \bigcup \mathbf{SW}$ is the set of vertices and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. A vertex $v_i \in \mathcal{V}$ represents a node in the architecture which is either an end system (ES) or a network switch (SW). An ES is either the source (talker) or the destination (listener) of an application flow, whereas an SW forwards the frames of flows. Nodes have input (ingress) and output (egress) ports. We denote the set of egress ports

TABLE 1. Summary of the notation.

Notation	Definition
$\mathcal{G}$	Network Graph
ES	End-System
SW	Network switch
$\nu_i \in \mathcal{V}$	Network node
$\nu_i.d(c)$	Forwarding delay
$p_j \in \nu_i.P$	Egress port
$q_j \in p_i.Q$	Priority queue
$\epsilon_{i,j} \in \mathcal{E}$	Link
$\epsilon_{i,j}.s$	Link speed
$\epsilon_{i,j}.d(c)$	Link delay
$\epsilon_{i,j}.mt$	Link macrotick
$r_i \in \mathcal{R}$	Route
$\lvert r_i \rvert$	Number of links in a route
$s_i \in \mathcal{S}$	Flow
$s_i.p$	Flow priority
$s_i.c$	Flow size
$s_i.t$	Flow period
$s_i.d$	Flow deadline
$\lvert s_i \rvert$	Number of flow instances
$f^k_{i,m}$	Frame
$f^k_{i,m}.\phi$	Frame offset
$f^k_{i,m}.l$	Frame length
$\gamma_i \in \Gamma$	Control application
$\gamma_i.K$	Control function
$\gamma_i.\mathcal{I}$	Set of input flows
$\gamma_i.\mathcal{O}$	Set of output flows

of a node with $v_i.P$. A port $p_j \in v_i.P$ is linked to at most one other node. The set of edges $\mathcal{E}$ represents bi-directional full-duplex physical links. Thus, a full-duplex link between the nodes v_i and v_j is denoted with both $\epsilon_{i,j} \in \mathcal{E}$ and $\epsilon_{j,i} \in \mathcal{E}$; a link is attached to one port of the node v_i and one port of the node v_j.

Each link $\epsilon_{i,j}$ is characterized by the tuple $\langle s, d, mt \rangle$ denoting the speed of the link in Mbit/s, the transmission delay function of the link and the macrotick, i.e., time granularity of an event for the link, in μs. The transmission delay function of a frame on a link $\epsilon_{i,j}.d(size)$ is calculated based the frame's size and the link speed. For example, transmitting a maximum transmission unit (MTU)-sized IEEE 802.1Q Ethernet frame of 1,542 bytes on a 1 Gbit/s link would take 12.33 μs. The function d is a notation used in the constraints in Sect. V and it is attached to the link concept, i.e., $\epsilon.d(size)$ means $d(\epsilon.s, size)$.

A route $r_i \in \mathcal{R}$, where $\mathcal{R}$ is a set of routes, is an ordered list of links, starting with a link originating from a talker ES, and ending with a link to a listener ES. The number of links in the route r_i is denoted with $|r_i|$, and it starts from 2 since we assume there is at least one SW in the route. We define the function $\mathbf{u} : \mathcal{R} \times \mathbb{N}_0 \rightarrow \mathcal{E}$ to capture the jth link of the route r_i.

An architecture model with three ESes two SWs is presented in Fig. 2, where the thick lines are physical links. We also show in the figure examples on how the notation is used, e.g., for a link tuple, ports, and routes.

B. TSN SWITCH MODEL

In the introduction we have motivated the use of TSN and the choice of traffic type for application messages, i.e., Scheduled Traffic (ST) that is being sent based on schedule tables in

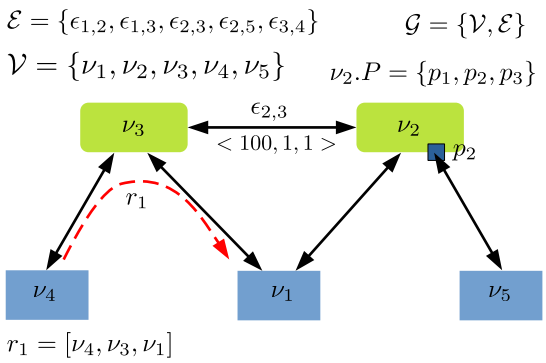

FIGURE 2. Architecture model example: the blue boxes are end systems, the green boxes are switches and the thick arrows are full-duplex physical links.

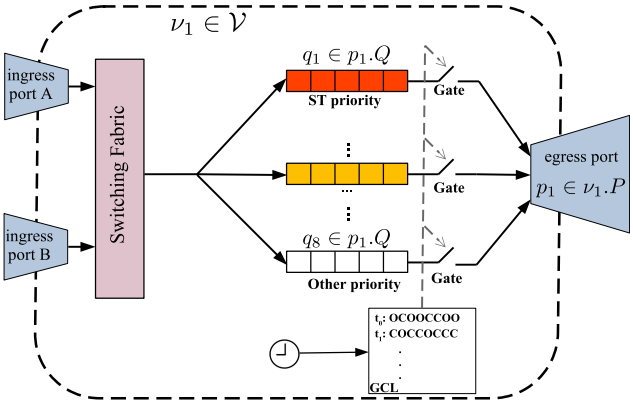

FIGURE 3. TSN switch internals.

the switches using the IEEE 802.1Qbv "Enhancements for Scheduled Traffic" amendment. Here we model the details of a TSN switch needed to formulate our problem. For further details on how TSN works, the reader is directed to the respective standards.

A TSN switch consists of ingress ports, a switching fabric, priority queues, gates, a Gate Control List (GCL) and egress ports, see Fig. 3. The switching fabric receives flows from the ingress ports and forwards each flow to the egress port p_i, according to the frame's route. The egress port which has a set of eight priority queues $p_i.Q$ (according to the IEEE 802.1Q standard [19]), stores the flow in a relevant priority queue $q_j \in p_i.Q$ in First-In-First-Out (FIFO) order. A subset of the priority queues are used for the ST traffic and the remaining queues are used for the less critical traffic, similar to [20]. Each frame has a Priority Code Point (PCP) field in the frame header that specifies the priority.

According to the 802.1Qbv standard, transmission of traffic from each queue is regulated by an associated gate which opens and closes based on a predefined GCL which contains the opening and closing time of the switch gates. Queued flows in a queue can be transmitted when a gate is open and cannot be transmitted when gate is closed. In this paper we assume that the GCLs are deterministic, i.e., the flows are isolated from each other: Only the frames of one of the flows are present in a queue at a time, see [20] for details.

Related work has ignored the *forwarding delay* that a frame experiences in a switch, which is the time it takes a frame to get from the input (ingress) port to the queue of the output (egress) port. This transmission delay is not related to the time the frame spends in the queue.. However, since delays have an impact on QoC [21], we have decided to capture the forwarding delay in our model, and depends on the particular TSN switch implementation. Hence, we denote the forwarding delay with $\nu_i.d(c)$ which takes c (frame size in bytes) as the input and returns the time delay in μs. In the experiments we measured this delay for the TSN implementation reported in [22].

C. APPLICATION MODEL

An FCP hosts multiple applications of mixed-criticalities, e.g., critical control applications, real-time applications, and best effort applications. Applications are typically modeled as interacting periodic real-time tasks that exchange messages, see [17] for how application tasks can be modeled. In this paper we address the configuration of the TSN communication infrastructure, hence we focus on messages. Sect. III discusses how tasks and messages can be put together in a system-level configuration.

Our model consists of a set of applications, which can either be *control* applications, for which their QoC is important, or they can be *real-time* applications. Note that control applications are also real-time, but not all real-time applications are control applications. The set of control applications is denoted with Γ. The tasks of both control and real-time applications exchange messages, which, if they are on different ESes, are transmitted using flows. The set of all flows (also called streams) in the system—both control and real-time flows—are denoted with $\mathcal{S}$.

Each flow $s_i \in \mathcal{S}$ is responsible for sending the frames that encapsulate the data from an application message and it is characterized by the tuple $\langle p, c, t, d \rangle$ denoting the priority, the size in bytes, the period in milliseconds and the flow deadline, i.e., the maximum allowed end-to-end delay in milliseconds. The priority of a flow is in the range from 0 to 7, where 0 is the highest priority concerning the eight priority queues of a switch egress port).

As mentioned, flows are periodic and may have different periods. We define the *hyperperiod* as the least common multiple of the periods of all flows. Depending on its period, the frames of a flow will have to be transmitted multiple times within a hyperperiod, and we refer to each such transmission as an *instance* of a flow. The number of instances for a flow s_i is denoted with $|s_i|$, and is derived from the period of the flow t and the hyperperiod. For example, for three flows with the periods of 4, 5 and 3 ms, the hyperperiod would be 60 ms and the flows will have 15, 12 and 20 instances respectively.

Each flow s_i is transmitted via a route r_j which is captured by the function $\mathbf{z} : \mathcal{S} \rightarrow \mathcal{R}$ that maps the flows to the routes. We assume that each flow is associated to only one route but several flows may share the same route. We also assume that the flows are unicast, i.e., there is only one listener for a flow. Our model can be easily be extended

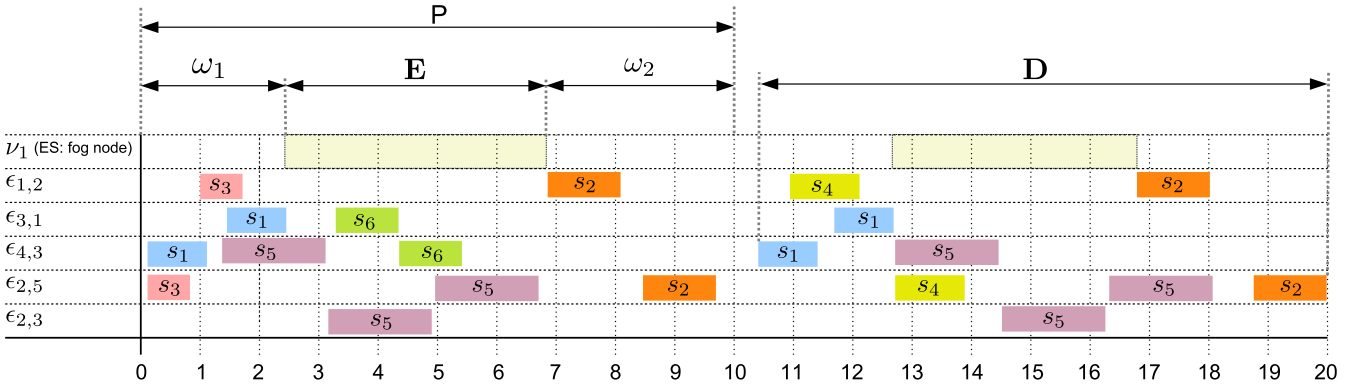

FIGURE 4. Example solution: ν_1 is a fog node and runs the control function during the execution slice denoted with E. The flow s_1 is the input flow and the flow s_2 is the output flow. The control application's period denoted with P, is 10 ms (s_1 and s_2 have the same period).

TABLE 2. Application example with six flows and two control applications.

Flow	Type & Details	priority p	size c (bytes)	period t (ms)	deadline d (ms)	routing
s_1	$\gamma_1.\mathcal{I}^1$	0	120	10	10	$\langle\epsilon_{4,3},\epsilon_{3,1}\rangle$
s_2	$\gamma_1.\mathcal{O}^1$	0	240	10	10	$\langle\epsilon_{1,2},\epsilon_{2,5}\rangle$
s_3	$\gamma_2.\mathcal{I}^1$	1	90	20	20	$\langle\epsilon_{5,2},\epsilon_{2,1}\rangle$
s_4	$\gamma_2.\mathcal{O}^1$	1	180	20	20	$\langle\epsilon_{1,2},\epsilon_{2,5}\rangle$
s_5	RT	0	450	12	12	$\langle\epsilon_{4,3}.\epsilon_{3,2},\epsilon_{2,5}\rangle$
s_6	RT	1	160	16	16	$\langle\epsilon_{1,3},\epsilon_{3,4}\rangle$

[1] Application transfer functions are $\gamma_1.K = \frac{100}{s^2+200}$ and $\gamma_2.K = \frac{250}{s^2+s}$.

to handle multicast flows, i.e., that have multiple listeners, by adding each talker-listener pair as a stand-alone flow with additional constraints. We assume that the routes are fixed and given. Determining routing in TSN is an orthogonal problem with scheduling. Researchers have shown how to integrate routing with scheduling [23] and have concluded that most shortest-path routing is appropriate in most network topologies, with the exception of mesh networks that have a lot of redundant links. Our system model, including the Constraint Programming model from Sect. V-B can be extended to include routing optimization, if needed.

We define a frame for each instance $1 \leq m \leq |s_i|$ of the flow s_i and on each link $1 \leq k \leq |r_j|$ of the route r_j, and denote it with $f^k_{i,m}$. A frame $f^k_{i,m}$ is associated with the tuple $\langle\phi, l\rangle$ denoting the start time of the frame (offset ϕ) and its duration (length l).

A control application $\gamma_i \in \Gamma$ is characterized by the tuple $\langle K, \mathcal{I}, \mathcal{O}\rangle$ denoting the control transfer function, the set of input flows, and the set of output flows. The control transfer function $\gamma_i.K$ captures the control law of the application, see Sect. IV for more details. The set of input flows $\gamma_i.\mathcal{I}$ is a subset of $\mathcal{S}$ which represents the control I/O flows that are generated by sensors (i.e, ESes in the network) and deliver data to the control application running on an ES. The set of output flows $\gamma_i.\mathcal{O}$ is a subset of $\mathcal{S}$ which represents the control I/O flows that are generated by control function running on an ES and deliver data to actuators (i.e, ES on the network).

III. PROBLEM FORMULATION

We formulate the problem as follows: Given (1) the set of all flows $\mathcal{S}$ in the system, for both the control and the real-time applications, (2) the details of the control applications Γ, (3) the network graph $\mathcal{G}$, and (4) a set of routes $\mathcal{R}$, we are interested in synthesizing the GCLs in the network such that (a) all the flows in the system are schedulable (their deadlines are satisfied) and (b) the QoC of control applications, as defined in Sect. IV-C, is maximized. Synthesizing the GCLs is equivalent to determining (i) the frames' offsets $f^k_{i,m}.\phi$, and (ii) the frames' length $f^k_{i,m}.l$. An example solution, considering the network from Fig. 2 and the flows from Table 2 is presented in Fig. 4. The solution is depicted as a Gantt chart where the rows are the resources (links) and the rectangles labeled with the flow names s_i depict the frames' offsets and lengths.

As discussed, the network configuration problem we address in this paper is orthogonal to the problem of configuring the tasks, e.g., deciding their mapping to the cores of an ES and their scheduling. Researchers have proposed several ways of putting together the schedules for tasks and messages in a global system configuration, e.g., by combining the formulation of their scheduling problems [24] or by iteratively integrating the task and message scheduling. The solution presented in this paper for flows can be combined with the formulation for tasks from [17]. In addition, to support the integration of the GCLs that we determine with tasks schedules derived separately, we maximize the time duration where tasks have to execute, denoted with E in Fig. 4, see see Sect. V-C for its definition.

IV. CONTROL THEORY

This section gives the essentials of the theory needed for the calculation of the QoC. We start with the definition of an Feedback Control Systems (FCS) in Sect. IV-A where the mathematical representation of a plant and the associated controller, and also the control design principle are described. Afterwards, we continue with the model we used for implementing a control application and a brief definition of the control performance and the effect of timing on it, in Sect. IV-B. Finally, we define in Sect. IV-C the QoC and present the approach we use in this work for calculating it.

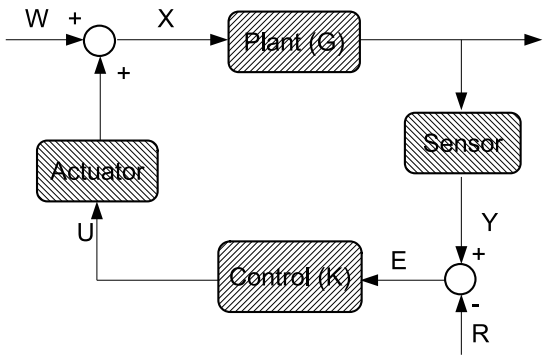

FIGURE 5. A simple FCS.

A. FEEDBACK CONTROL SYSTEMS AND CONTROL DESIGN

A dynamical system around an equilibrium point is modeled as a mathematical relation between its inputs and outputs, and described with a transfer function [25]. The transfer function, commonly called *Plant*, is defined in the form of

$$Y(s) = G(s) \times X(s), \tag{1}$$

where $Y(s)$ is the outputs, $X(s)$ is the inputs, and $G(s)$ is the transfer function, all defined in the frequency domain. An FCS, or alternatively a control application, uses sensors to sample the plant's outputs $Y(s)$, calculates the deviation $E(s)$ from the desired output $R(s)$ and uses the control function $K(s)$ to generate the control signal $U(s)$ which is applied by actuators. In this paper, we assume that the desired output $R(s)$ is zero which results in $E(s) = Y(s)$.

The control function $K(s)$ defines the mathematical relation between the deviation $E(s)$ of the plant feedback from desired output, and the control signal $U(s)$. A simple FCS is depicted in Fig. 5, where $W(s)$ are the disturbances applied to plant inputs.

An FCS is implemented as a periodic real-time application running on a FCP whose period depends on the system plant $G(s)$. The shorter the period, the faster the controller is able to respond to the disturbances and the more computational power is required (which is a bottleneck on real-times systems where the resources are constrained). To this end, while designing an FCS choosing the right application period is an optimization problem. It is common to choose the period based on a rule of thumb which determines the period based on the bandwidth of the closed-loop system [26]. On the other hand, choosing an appropriate control law to be implemented in the control function $K(s)$ has an impact on the resources needed for the calculation and the its response to the disturbances. Several control laws are proposed in the literature for control functions [25].

B. MODELING AND TIMING OF FEEDBACK CONTROL SYSTEMS

The implementation of an FCS consists of three periodic events: (i) receiving the inputs data from sensors, (ii) calculating the control signal with control function $K(s)$, and (iii) sending the control signal data to actuators that apply the signal to the plant. Without the loss of generality, we assume that each FCS receives the input from exactly one sensor and sends signal data to exactly one actuator. We also assume that the three periodic events have the same period.

We map our FCS model to the control application model described in Sect. II-C as follows: A control application γ_i is an FCS that has the control function $\gamma_i.K$, equivalent to $K(s)$, running on the node v_j (which is an ES) in the network $\mathcal{G}$. The associated sensor is also an ES node that transmit a period network flow $s_m \in \gamma_i.\mathcal{I}$ to the node v_j as the destination via TSN. The generated control signal $U(s)$ is also a period network flow $s_n \in \gamma_i.\mathcal{O}$ transmitted from the node v_j to the associated actuator which is also an ES. To this end, the set of input flows $\gamma_i.\mathcal{I}$ and the set of output flows $\gamma_i.\mathcal{O}$ both have only one unique member which are s_m and s_n respectively.

Concerning our FCS model, the control function $\gamma_i.K$ is ready for execution when its input is arrived, i.e. the node v_j receives the input flow s_m; and produces the control signal s_n when it terminates. Thus the control signal s_n needs to be transmitted after the reception of the input signal s_m and execution of the control function $\gamma_i.K$. We formulate this constraint in Sect. V-B.

While designing an FCS, for finding the suitable control law and tuning it, several parameters such as the damping ratio, the phase margin and the gain margin (see [25] for more details) have to be determined. These parameters affect the accuracy and rapidity of the FCS which is called control performance, in opposite directions. The performance of an FCS is associated with its rise-time T_{rise}, peak-time T_{peak}, settling-time T_{settling} and steady state error.

The rise-time T_{rise} is defined as the time takes for the output response to reach 90% of the input value. The rise-time shows how fast the controller can react to the disturbances exerted to the dynamical system. The peak response is defined as highest output response the controller reached before the desired value. The peak plays an important role in the robustness of the controller against disturbances. The settling-time $T_{settling}$ is defined as the time takes for the output response to reach 98% of the input value. The settling-time shows how fast the controller can reach to the desired state. The steady-state error shows the minimum deviation of the controller output response from the desired state. It shows the accuracy of the controller. Fig. 6 shows the step-response of a sample control loop where these associated parameters are depicted.

Furthermore, for a given FCS whose design parameters are determined, the control performance changes in runtime due to the discrete time nature of the real-time systems. Ideally, all three events of an FCS should execute with the shortest delay between the events and without timing variations (jitter) as well. A time delay decreases the phase margin of the FCS leading to worse control performance. Jitter, i.e. the deviation from the periodic timing of an event, also negatively impacts the control performance.

We assume that the time delay and jitter apply only to the event of network message transmission, while the execution of the control application is ignored in this paper, and only

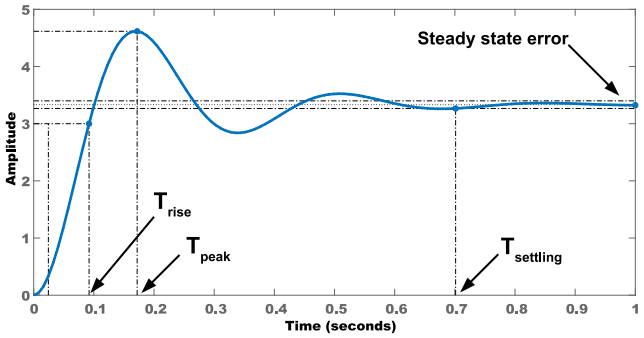

FIGURE 6. Step response of a sample control loop [17].

addressed as the required time interval needed between the reception and transmission of the input and output flows. We also consider the input-output jitter of the control application which is the maximum deviation of the worst-case delay between the sensors' sampling and the actuators actuation covering the timing of communication links from and to sensors and actuators.

C. QUALITY OF CONTROL

In real-time systems where control applications are running, preserving QoC (which is used interchangeably to mean ''control performance'') is a necessity. The QoC can be captured in a cost function which can also be used to evaluate the performance of the controller. A common choice is to use a quadratic cost function of the form

$$J = \int_0^\infty \left(x^T(t) Q_1 x(t) + u^T(t) Q_2 u(t) \right) dt, \qquad (2)$$

where the weighting matrices Q_1 and Q_2 tell how much deviations in the different states and the control input should be penalized. A larger value of such as control performance cost function means worse QoC and typically increasing the settling time, the steady state error and closer peak time to rise time of the system.

The value of cost J depends on several criteria such as Input-Output jitter of a control application as well as the end-to-end response of the control application (the delay between sampling and actuation). Generally, the control performance is degraded when the end-to-end response is more than what the control application is designed for or when the control application experiences Input-Output jitter in each iteration, see [27] for more details. The amount of each criterion's impact depends on the control function. To this end, the calculation of the QoC is possible via a simulation of the control function behavior. Tools such as Jitterbug [28], JitterTime [27] and TrueTime [29] are proposed to simulate the control function behavior. Jitterbug can calculate the QoC based on the fixed or random jitter applied to inputs and outputs of a control function. It can also be used to design controllers concerning the stability margin of the control function. JitterTime can calculate the QoC based on the inputs and outputs schedules as well as the control task schedules. Also, it can be employed to analyze the sensitivity of a control function to delays and jitter. TrueTime can simulate the execution of a control application based on a given schedule tables making the analysis of the control output possible. Thus, we employed JitterTime to calculate the QoC with the same cost function as Eq. (2) in this paper.

JitterTime takes the sending and receiving time of sensor and actuator flows which can be captured from GCLs, and simulates the behaviour of a control application with the given timing of control application's inputs and outputs. More information about the inner workings of JitterTime and its use cases can be found in [27].

V. CONSTRAINT PROGRAMMING

The communication scheduling problem as a decision problem has been proved to be NP-complete in the strong sense [30]. To this end, we propose an optimization strategy called Control-Aware Communication Scheduling Strategy (*CACSS*), based on a CP formulation that uses search heuristics inside the CP solver.

As shown in Fig. 7, *CACSS* takes as the inputs the architecture and application models and outputs a set of the best solutions found during search. As mentioned, *CACSS* is based on a CP formulation (the ''CP Solver'' box) in the figure. CP is a declarative programming paradigm that has been widely used to solve a variety of optimization problems such as scheduling, routing, and resource allocations. With CP, a problem is modeled through a set of variables and a set of constraints, see the ''CP model'' box. Each variable has a finite set of values, called domain, that can be assigned to it (see Sect. V-A). Constraints restrict the variables' domains by bounding them to a range of values and defining relations between the domains of different variables.

CACSS visits solutions that satisfy the constraints defined in Sect. V-B and evaluates them using the objective function defined in Sect. V-C to check if the solution is an *improving* solution, i.e., better than the best solutions found so far. Ideally, for the QoC calculation, the objective function should use JitterTime. However, tools such as JitterTime and Jitterbug use time consuming *simulations* of the control application behavior, and hence they cannot be integrated into a CP solver since the search will not scale. Thus, we propose a novel *analytical model* for the QoC evaluation within the CP formulation, see Sect. V-C. Every time the CP solver finds an *improving* visited solution (the ''New Solution'' box), we call JitterTime (the ''Jitter Time'' box) calculating the simulation-based accurate QoC value. By default, the CP solver systematically performs an exhaustive search by exploring all the possibilities of assigning different values to the variables. However, such a search is intractable for NP-complete problems, therefore we instead employ a metaheuristic search, see Sect. V-D.

A. CP MODEL

We define two sets of decision variables for the CP model, which are associated with the frame offsets and the frame lengths respectively. Each decision variable is associated with

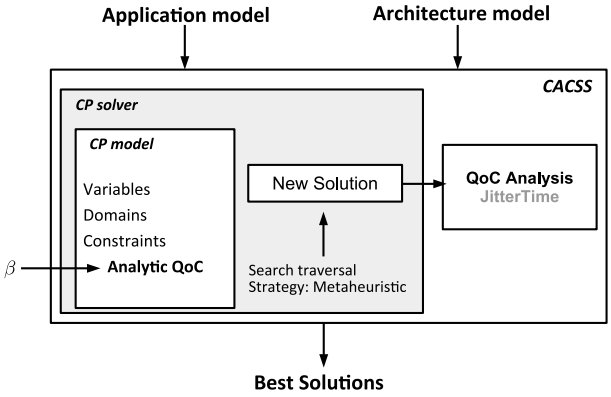

FIGURE 7. Overview of *CACSS*.

a domain from which the CP solver decides the variable's value. The decision variables and their domain are defined by

$$\begin{aligned}
&\forall s_i \in \mathcal{S}, \forall m \in [1, .., |s_i|], \forall k \in [1, .., |r_j|],\\
&r_j = \mathbf{z}(s_i), \epsilon_{v,w} = \mathbf{u}(r_j, k) :\\
&f^k_{i,m}.l = \frac{s_i.c}{\epsilon_{v,w}.s \times \epsilon_{v,w}.mt}\\
&0 \leq f^k_{i,m}.\phi \leq (\frac{s_i.t}{\epsilon_{v,w}.mt} - f^k_{i,m}.l) \qquad (3)
\end{aligned}$$

where the domain of the frame lengths contains exactly one element, i.e. the CP solver initially decides the values of frame length variables.

B. CONSTRAINTS

We define five constraints that regulate the network traffic and relates the domain of the CP variables. CP only finds the feasible solutions, i.e. all the constraints are met.

The *Link Overlap constraint* imposes the restriction on the solution to not allow a physical link to transmit more than one frame at a time, which is equivalent to avoid sharing a physical link with two frames at any time. The constraint is defined in Eq. (4).

$$\begin{aligned}
&\forall s_i, s_j \in \mathcal{S}, i \neq j, \forall m \in [1, .., |s_i|], \forall n \in [1, .., |s_j|],\\
&r_o = \mathbf{z}(s_i), \forall k \in [1, .., |r_o|], \quad r_p = \mathbf{z}(s_j), \forall l \in [1, .., |r_p|],\\
&\epsilon_{v,w} = \mathbf{u}(r_o, k) = \mathbf{u}(r_p, l) :\\
&(f^k_{i,m}.\phi + m \times \frac{s_i.t}{\epsilon_{v,w}.mt} \geq f^l_{j,n}.\phi + n \times \frac{s_j.t}{\epsilon_{v,w}.mt} + f^l_{j,n}.l) \vee\\
&(f^l_{j,n}.\phi + n \times \frac{s_j.t}{\epsilon_{v,w}.mt} \geq f^k_{i,m}.\phi + m \times \frac{s_i.t}{\epsilon_{v,w}.mt} + f^k_{i,m}.l). \qquad (4)
\end{aligned}$$

The *Route constraint* enforces the ordered propagation of a frame concerning its associated route from its talker all the way to its listener. The constraint also enforces that forwarding a frame from a node starts after it has completely arrived at the reception of the node concerning the propagation delay. We define the constraint in Eq. (5) where δ is the network precision that is the worst-case difference between the nodes clock in the network according to the 802.1AS clock synchronization mechanism [15].

$$\begin{aligned}
&\forall s_i \in \mathcal{S}, \forall m \in [1, .., |s_i|], \forall k \in [1, .., |r_j|),\\
&r_j = \mathbf{z}(s_i), \epsilon_{v,w} = \mathbf{u}(r_j, k), \epsilon_{w,x} = \mathbf{u}(r_j, (k+1)),\\
&\Delta = \epsilon_{v,w}.d(s_i.c) + \nu_w.d(s_i.c) + \delta :\\
&f^{k+1}_{i,m}.\phi \times \epsilon_{w,x}.mt \geq (f^k_{i,m}.\phi + f^k_{i,m}.l) \times \epsilon_{v,w}.mt + \Delta. \qquad (5)
\end{aligned}$$

We define the *Isolation constraint* in Eq. (6) to avoid displacement of frames in different switch queues. The constraint imposes the restriction on any two same-priority frames on the same link not to arrive at the ingress port of a switch simultaneously. In another word, either a frame is received after or before any other frame on the same link, or the different priority frames on the same link are received at the same time. This constraint enforces the order of frame transmission in the switch schedules, see [20] for more details. In Eq. (6), δ represents the network precision.

$$\begin{aligned}
&\forall s_i, s_j \in \mathcal{S}, i \neq j, \forall m \in [1, .., |s_i|], \forall n \in [1, .., |s_j|],\\
&r_o = \mathbf{z}(s_i), \forall k \in (1, .., |r_o|], \quad r_p = \mathbf{z}(s_j), \forall l \in (1, .., |r_p|],\\
&\epsilon_{v,w} = \mathbf{u}(r_o, k), \epsilon_{a,b} = \mathbf{u}(r_p, l),\\
&\epsilon_{x,v} = \mathbf{u}(r_o, k-1), \epsilon_{y,a} = \mathbf{u}(r_p, l-1) :\\
&((f^k_{i,m}.\phi \times \epsilon_{v,w}.mt + m \times s_i.t + \delta\\
&\leq f^{l-1}_{j,n}.\phi \times \epsilon_{y,a}.mt + n \times s_j.t + \epsilon_{y,a}.d(s_j.c))\\
&\vee (f^l_{j,n}.\phi \times \epsilon_{v,w}.mt + n \times s_j.t + \delta \leq\\
&f^{k-1}_{i,m}.\phi \times \epsilon_{x,v}.mt + m \times s_i.t + \epsilon_{x,v}.d(s_i.c)))\\
&\vee (s_i.p \neq s_j.p). \qquad (6)
\end{aligned}$$

The *Deadline constraint* defined in Eq. (7) imposes the restriction that a flow is received by its listener within its deadline. This constraint is equivalent to that the time interval between the scheduled transmission of a stream from its talker and the reception of it by the listener is smaller than its deadline.

$$\begin{aligned}
&\forall s_i \in \mathcal{S}, \forall m \in [1, .., |s_i|], r_j = \mathbf{z}(s_i),\\
&\epsilon_{a,b} = \mathbf{u}(r_j, 1), \epsilon_{y,z} = \mathbf{u}(r_j, |r_j|) :\\
&f^1_{i,m}.\phi \times \epsilon_{a,b}.mt + s_i.d \geq \epsilon_{y,z}.mt \times (f^{|r_j|}_{i,m}.\phi + f^{|r_j|}_{i,m}.l). \qquad (7)
\end{aligned}$$

The *Control Precedence constraint* enforces every instance of a control application's output flows to be scheduled for transmission after the complete reception of the same-instance input flows at the listener. Thus, the control application's output flows are transmitted from the talker node after the execution of the control function is terminated which needs the complete reception of the input flows. The constraint is defined in Eq. (8).

$$\begin{aligned}
&\forall \gamma_i \in \Gamma, \forall s_j \in \gamma_i.\mathcal{I}, \forall s_k \in \gamma_i.\mathcal{O},\\
&\forall m \in [1, .., |s_j|], \forall n \in [1, .., |s_k|],\\
&r_o = \mathbf{z}(s_j), r_p = \mathbf{z}(s_k),\\
&\epsilon_{a,b} = \mathbf{u}(r_o, |r_o|), \epsilon_{b,z} = \mathbf{u}(r_p, 1),\\
&\Delta = \epsilon_{a,b}.d(s_j.c) + \nu_b.d(s_j.c) + \delta :\\
&(f^{|r_o|}_{j,m}.\phi + f^{|r_o|}_{j,m}.l) \times \epsilon_{a,b}.mt + \Delta \geq \epsilon_{b,z}.mt \times f^1_{k,m}.\phi \qquad (8)
\end{aligned}$$

C. ANALYTICAL QoC CP MODEL AND OBJECTIVE FUNCTION

The CP solver propagates the constraints all over the search spaces and removes the unfeasible solutions (which do not satisfy the constraints) from the search space that results in the creation of the solution space. Afterwards, the CP solver picks the first solution from the solution space and determines the value of the objective function for the solution. The CP solver searches for better solutions in terms of the objective function until no such solutions can be been found.

In this work, we are interested in finding the solutions which have better QoC. Since calculating the QoC needs a simulation of the control application's behavior, the integration of QoC calculation tools such as JitterTime in the CP model is impossible due to their runtime. Thus, we propose using an analytical model for QoC as the objective function in the CP model, which aims to drive the search to solutions that are as close as possible (in terms of the QoC value obtained with JitterTime simulations) to solutions obtained if JitterTime would be used as an objective function for the search.

Our proposed analytical model captures within the CP formulation: (i) minimum jitter for end-to-end input-output flows, (ii) maximum delay between reception of the input flow and transmission of the output flow (which is equivalent to minimum input flow delay and minimum output flow delay), denoted with E and called task execution interval, and (iii) minimum jitter for the task execution interval.

Let us illustrate these aspects using the example in Fig. 4 where we have a Gantt chart for the execution of an example control loop depicting components of our analytical model. In this toy example, we have a control application γ_1 which has s_1 as the input flow and s_2 as the output flow. The application's control function $\gamma_1.K$ is running on the node ν_1. The flow s_1 is transmitted from the sensor node ν_4 and routed via the switch ν_3 to the node ν_1 and has the same period as the control application, denoted with P in the figure. The flow s_2 is transmitted from the node ν_1 and routed via the switch ν_2 to the actuator node ν_5 and also has the same period as the control application.

The node ν_1 runs the control function once its input flow s_1 arrives and transmits the flow s_2 on the terminal of the control function. Thus, the larger the task execution interval$E$, the more probable that the control function implemented as tasks are scheduled for execution on the node ν_1. Since we need to define the CP objective function to be minimized, and the control application has the known period P, the objective would be to minimize the ω_1 and ω_2 which are, respectively, the input flow and the output flow end-to-end delay. Furthermore, we are interested in minimizing the variation of the task execution interval $\mathcal{E}$ which results in more possibility of the control function's schedulability. This is also formulated as minimizing the input and output flows jitter.

Additionally, minimizing ω_1 and ω_2 and their variation positively impacts on the QoC, since the control function receives the plant's sampling faster and without variation and the control signal is applied to the plant faster and without variation as well. However, the control function implemented as tasks could be scheduled for execution anywhere in the execution slice, but because of the jitter-free and short-delay input output the negative side of the task scheduling is compensable.

We define the QoC analytical function Ω in Eq. (9), where the terms ω_1 captures input flow delay, ω_2 captures output flow delay, ω_3 captures input flow jitter, ω_4 captures output flow jitter and ω_5 captures E jitter. The range of all the ω terms is from 0 for no delay/jitter to 1 for a delay/jitter equal to the control application's period. The delay and jitter trade-off is controlled by the weight β which can direct the search towards either optimized delay or optimized jitter, concerning the type of the control applications. A larger β value drives the search towards smaller jitter. The β value can be determined by analyzing using JitterTime the behavior of the control function regarding the sensitivity to jitter and delay. JitterTime simulates the behavior of a control function with a given delay and jitter values. Hence, given different delay and jitter, JitterTime is capable of determine the sensitivity ratio. Thus, we use JitterTime for analyzing the sensitivity and determining the β value for a control function.

$$
\begin{aligned}
&\forall \gamma_i \in \Gamma, \forall s_j \in \gamma_i.\mathcal{I}, \forall s_k \in \gamma_i.\mathcal{O},\\
&\forall m, q \in [1, .., |s_j|], \forall n, u \in [1, .., |s_k|],\\
&r_o = \mathbf{z}(s_j), r_p = \mathbf{z}(s_k),\\
&\epsilon_{a,b} = \mathbf{u}(r_o, |r_o|), \epsilon_{e,f} = \mathbf{u}(r_o, 1),\\
&\epsilon_{c,g} = \mathbf{u}(r_p, |r_p|), \epsilon_{b,z} = \mathbf{u}(r_p, 1) :\\
&\omega_1 = \sum \frac{f_{j,m}^{|r_o|}.\phi \times \epsilon_{a,b}.mt}{s_j.t}\\
&\omega_2 = \sum \frac{s_k.t - f_{k,n}^{1}.\phi \times \epsilon_{b,z}.mt}{s_k.t}\\
&\omega_3 = \sum \frac{|(f_{j,m}^{|r_o|}.\phi - f_{j,q}^{|r_o|}.\phi) \times \epsilon_{a,b}.mt + (m-q) \times s_j.t|}{s_j.t}\\
&\omega_4 = \sum \frac{|(f_{k,n}^{1}.\phi - f_{k,u}^{1}.\phi) \times \epsilon_{b,z}.mt + (n-u) \times s_k.t|}{s_k.t}\\
&\omega_5 = \sum \frac{|(f_{k,m}^{|r_p|}.\phi - f_{k,q}^{|r_p|}.\phi) \times \epsilon_{c,g}.mt +}{}\\
&\frac{(f_{j,q}^{1}.\phi - f_{j,m}^{1}.\phi) \times \epsilon_{e,f}.mt + (m-q) \times s_j.t|}{s_j.t}\\
&\Omega = \omega_1 + \omega_2 + \beta \times (\omega_3 + \omega_4 + \omega_5)
\end{aligned}
\tag{9}
$$

D. SEARCH STRATEGY

In this work we used the Google OR-Tools [31] CP solver. We configured this solver to use a *metaheuristic* as the search strategy. A search strategy specifies the order of selecting the CP model variables for assignment and the order of selecting the values from the domain of a variable. The metaheuristic strategy does not guarantee optimality but it is effective in finding a good quality solutions in a reasonable time.

We used the same metaheuristic strategy as [16] based on a Tabu Search metaheuristic algorithm [32], which aims to avoid the search process being trapped in a local optimum by increasing diversification and intensification of the search. We apply the metaheuristic strategy to the set of offset variables $f^k_{i,m}.\phi$ that represents control-I/O flows. In this strategy, once a control application is scheduled with the respective minimum objective value, it is treated as keep variables whose values should not be changed. We also used *SolveOnce* strategy for the set of length variables $f^k_{i,m}.l$.

VI. EVALUATION

The structure of this section is as follows: we first describe our test setup and the test cases we used for evaluation in Sect. VI-A followed by comparing our proposed Control-Aware Communication Scheduling Strategy (*CACSS*) with the related work in Sect. VI-B. Afterwards, we evaluate our proposed method on the synthetic test cases in Sect. VI-C. In Sect. VI-D we evaluate *CACSS* on a realistic test case. We also validate the generated GCLs using the *OMNET++* simulator in Sect. VI-E. Finally, we used the generated GCLs and validated them on a TSN hardware platform in Sect. VI-F.

A. TEST CASES AND SETUP

We implemented *CACSS* in Java using Google OR-Tools [31] as the CP solver and run it on a computer with an i9 CPU at 3.6 Ghz and 32 GB of RAM. We have considered a time limit for the CP solver of 10 to 100 minutes depending on the test case size. For the evaluation we set the macrotick, the network precision and the link speed to 1 μs, 0 μs and 100 Mbit/s, respectively.

We have generated thirteen synthesis test cases which all include control applications inspired from the industrial domain. The control applications have different control functions for controlling plants in the form of Eq. (10) where a and b are randomly chosen respectively from [50, 100, 150] and [100, 200, 300, 400]. We have used Jitterbug for designing the control function K with the LQG control law [28] as discussed in Sect. IV-C. The test case sizes are progressively increasing in number of ESes, SWs, and flows (and respectively control applications). The flows are generated randomly with various sizes to fit in single MTU-sized frames, various periods all in the form of 2^n ms, $n = \{0, 1, 2, 3, 4\}$, and various priorities. The details of the synthetic test cases are depicted in Table 3 where the sixth column shows the total number of flow frames.

$$G = \frac{a}{s^2 + b} \tag{10}$$

We have also considered a realistic test case, an autonomous mobile robot, called AMR. The AMR case consists of 27 flows varying in size between 100 and 1,500 bytes, with periods between 1 ms and 40 ms and deadlines smaller or equal to the respective periods. We used Jitterbug for designing the control functions from the plant in Eq. (10). The details of the realistic test case are shown in Table 6.

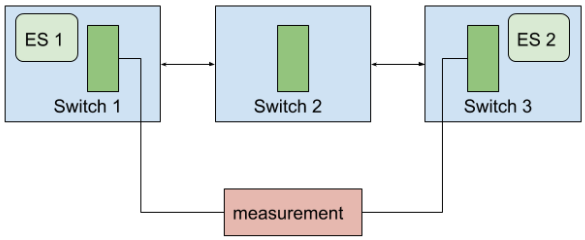

FIGURE 8. Schematics of the hardware platform.

Additionally, we generated three test cases for evaluating on a hardware platform. The generated GCLs are implemented on the platform and the end-to-end (E2E) delay—the time between sending a frame from it source to the time it arrives at its destination—and jitter of flows are measured. The details of the test cases are shown in Table 5. The hardware platform is presented in [22] and consists of three TSN switches that are connected in a daisy chain manner. The first and the last switches consist internal ESes. The links are full duplex with the speed of 1 Gbps and flows can be sent from both ESes. A schematic of the hardware platform is shown in Fig. 8 where the points for the measurement are marked.

B. COMPARISON WITH THE RELATED WORK

Let us first compare *qualitatively* the features of our *CACSS* with the approaches of the related work, i.e., (i) Zero-Jitter GCL (*ZJGCL*) proposed in [20] and (ii) Frame-to-Window GCL (*FWGCL*) proposed in [33]. Table 4 summarizes the model features, where the first column lists the feature compared. *CACSS* and *ZJGCL* consider scheduling of each individual flow frame and leads to zero jitter solutions under the jitter-optimized condition, whereas *FWGCL* schedules "windows" which may contain several frames, reducing thus the size of GCLs at the expense of introducing jitter. Considering flow frames for scheduling, *CACSS* and *ZJGCL* both enforce "frame isolation" that results in frames with zero jitter, see [20] for a discussion of the need for frame isolation to create deterministic GCLs. All the three approaches consider network precision.

The main advantage of *CACSS* over the related work is the modeling of control applications, i.e., the precedence constraints of input and output flows and the task execution interval. None of the related work considers the control application modeling, which makes the assessment of QoC impossible. To integrate the evaluation of control performance into the optimization, we have formulated the QoC analytically capturing the minimization of the input-output and execution jitter of control applications and also leaving enough time space for the control functions to be executed. In addition, the *CACSS* also considers a model for forwarding delay of SWs, which makes the schedules more accurate considering a TSN hardware implementation.

We have also performed a *quantitative* comparison our proposed method *CACSS* with the *ZJGCL* approach from the related work. Note that a comparison between *ZJGCL* and *FWGCL* is provided in [33], and since *FWGCL* introduces

TABLE 3. Evaluation on the synthetic test cases.

#	Total no. of flows	Total no. of control apps	Total no. of SWs	Total no. of ESes	Total no. of frames	Ω	QoC for CACSS	QoC for ZJGCL	Runtime (ms)
1	8	1	2	6	53	66.1	0.862	1.335	132
2	12	1	2	6	68	66.0	0.860	1.415	138
3	14	2	2	6	60	77.4	0.959	1.409	147
4	8	1	3	6	77	110.2	1.367	1.985	170
5	16	3	4	8	89	67.1	0.872	1.605	621
6	24	3	5	10	171	67.3	0.881	1.821	1351
7	16	2	5	8	100	58.0	0.771	1.177	743
8	20	3	6	10	149	66.5	0.867	1.187	943
9	24	4	7	10	198	90.0	1.152	1.555	1465
10	30	4	7	10	244	90.0	1.153	1.661	1793
11	27	5	20	20	1770	151.0	1.889	2.957	6225
12	27	5	20	20	1834	151.0	1.897	3.805	9153
13	27	7	20	20	1770	149.6	1.865	3.411	3553

TABLE 4. Comparison of different communication scheduling mechanisms.

Item	ZJGCL	FWGCL	CACSS
Scheduling object	frame	window	frame
Frame isolation	Yes	No	Yes
Network precision	Yes	Yes	Yes
Control app. model	No	No	Yes
Forwarding delay	No	No	Yes
Point-to-Point Tunneling Protocol (PPTP) flows scheduling	No	No	Yes

scheduling flexibility at the expense of jitter, it will lead to worse control performance. Hence, due to this, and for space reasons, we have not compared against *FWGCL*. *ZJGCL* does not consider control performance, hence, in order to facilitate a comparison, we have reimplemented *ZJGCL* using a CP formulation and added constraints that enforce the construction of valid solutions, i.e., to schedule the output flows to be transmitted after the reception of the input flows and to be received close to their deadline (leaving enough space for execution of the control functions). The GCLs obtained with both *CACSS* and *ZJGCL* were then evaluated using JitterTime, which accurately measures the control performance of each solution. The evaluation results are depicted in columns 8 and 9 in Table 3. The results show that *CACSS* has generated schedules with significantly better QoC than *ZJGCL*. The average QoC for *ZJGCL* is 64% larger. *ZJGCL* schedules flows such that jitter becomes zero; this is useful but not sufficient for a good QoC value, which also depends on input-output jitter and input/output delay. In addition, our method also maximizes the task execution intervals, which support the integration of the resulted schedules with the schedules for tasks. In contrast, the *ZJGCL* GCLs will have to be drastically modified before they can be integrated with task schedules.

C. EVALUATION ON SYNTHETIC TEST CASES

We evaluated the performance of *CACSS* on the synthetic test cases from Table 3. Our solution has successfully scheduled all the test cases and the schedules have zero jitter. We first evaluate the runtime of our proposed solution. The solution

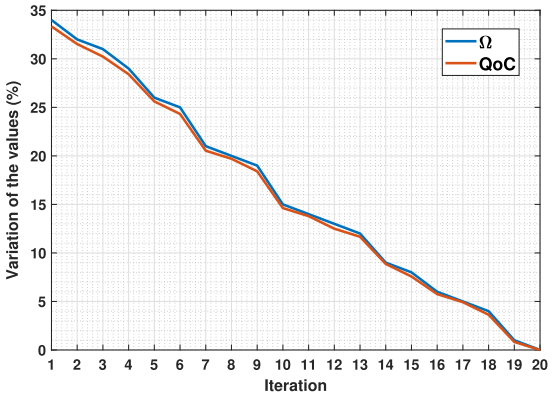

FIGURE 9. Comparison of analytical function Ω with QoC obtained by JitterTime for test case 5 from Table 3.

runtime in milliseconds for each test case is given in column 10 in Table 3. As depicted in the table, the runtime increases with the increase of the total number of frames, i.e., larger test cases. As mentioned, we have given a time limit to the solver, between 10 and 100 min., depending on the test case size. All runs have finished well before the time limits, which means that the *CACSS* was able to determine the optimal results in terms of the objective function value from Eq. (9). This shows that, using our analytical QoC model inside the CP formulation, we are able to solve large test cases in a reasonable time.

The columns 7 and 8 in Table 3 show the objective function value Ω Eq. (9) and QoC measured with JitterTime (which corresponds to the J value captured by Eq. (2)). The question is if driving the search with Ω, which is a "proxy" for QoC, as we do in *CACSS* is as good as driving the search with J, which is the actual QoC value. Hence, we were interested to determine if our analytical QoC model Ω is able to drive the search to solutions with good QoC. Thus, for a test case 5 from Table 3, we have replaced the fast analytical QoC model in the CP formulation with the simulation-based slow-but-accurate JitterTime QoC value. We have run *CACSS* for test case 5 with both setups, using Ω from Eq. (9) vs. the

TABLE 5. Details of the implemented-on-hardware test cases: Sizes are in bytes, Periods and deadlines are in μs.

Flow	Size (bytes)	Period (μs)	Deadline (μs)	Talker	Listener	Reported max. E2E delay (μs)	Measured max. E2E delay (μs)	Reported max. E2E jitter (μs)	Measured max. E2E jitter (μs)
Hardware Test Case 1									
1	400	600	2400	ES1	ES2	58	58.32	0	0.46
2	100	1200	2400	ES1	ES2	20	19.93	0	0.20
3	300	800	2400	ES2	ES1	46	46.23	0	1.87
4	60	2400	2400	ES2	ES1	16	15.82	0	0.06
Hardware Test Case 2									
1	400	1200	4800	ES1	ES2	58	58.34	0	0.49
2	100	2400	4800	ES1	ES2	20	19.89	0	0.14
3	300	1600	4800	ES2	ES1	46	46.27	0	1.82
4	60	4800	4800	ES2	ES1	16	15.82	0	0.06
Hardware Test Case 3									
1	400	2400	9600	ES1	ES2	58	58.33	0	0.49
2	100	4800	9600	ES1	ES2	20	19.88	0	0.12
3	300	3200	9600	ES2	ES1	46	46.26	0	1.80
4	60	9600	9600	ES2	ES1	16	15.82	0	0.05

TABLE 6. Evaluation on realistic test case: AMR consists 27 flows, 20 ESes, 20 SWs.

Test case	No. of control apps	Average E2E delay	Ω	Runtime	No. of frames
AMR	9	52	134	2348 ms	1452

QoC value J obtained with a call to JitterTime. The results are shown in Fig. 9, where we compare the two values (y-axis) during the search, i.e., during the iterations listed on the x-axis. On the y-axis we have the percentage deviation of Ω and J for their best respective values obtained at the end of the search; in the last iteration, the deviation is zero, because we have the best value for both of them. As we can see in the figure, our analytic model of QoC closely tracks the simulation-based model of QoC, which supports our hypothesis that the analytical QoC model Ω is a good proxy objective function for guiding the search.

D. EVALUATION ON A REALISTIC TEST CASE

We also evaluated *CACSS* on the autonomous mobile robot (AMR) realistic test case. The results of the evaluation are presented in Table 6, where column 2 shows the number of control applications. In the realistic test case, we assumed that the link speed is 1 Gbps. The *CACSS* has successfully scheduled all the flows in the test case and achieved a good QoC, which is captured by the objective function Ω (column 4 of the table).

E. OMNET++ VALIDATION

We have used the *OMNET++* simulator with the TSN NeSTiNg extension [34] to validate the generated GCLs, and also measured the average delay and jitter of the solutions. Our goal was to evaluate the correctness and the accuracy of our proposed solution within a realistic simulation environment.

The NeSTiNg extension of *OMNET++* ignores the forwarding delay, so to facilitate a fair comparison we updated our *CACSS* approach creating a variant that considers a zero forwarding delay (ZFD), and named it *CACSS*-ZFD.

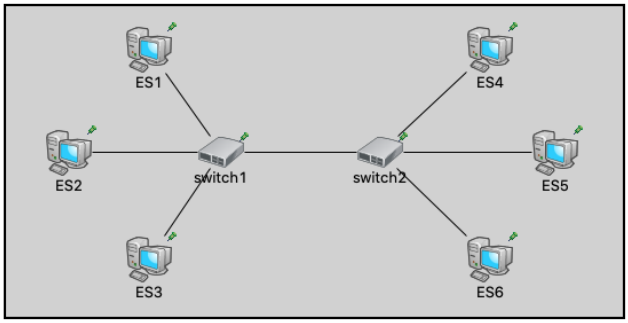

FIGURE 10. Implementation of TC1 in OMNET++.

TABLE 7. Simulation results of the synthetic test case 1 from Table 3.

Flow ID	Observed E2E delay in OMNET++ (μs)	Reported E2E delay in CACSS-ZFD(μs)
1	245	245
2	318	318
3	332	332
4	274	274
5	210	210
6	142	142
7	178	178
8	387	387

We took the synthetic test case 1 from Table 3, synthesized the GCLs with both *CACSS* and *CACSS*-ZFD. We simulated the schedules of all the synthetic and realistic test cases from Tables 3 and 6 using *OMNET++*. The schedules behave as expected and the delays we extract from the *OMNET++* simulations are identical with the values obtained by our *CACSS*. Let us provide more details for one of the test cases. Fig. 10 shows the architecture of the synthetic test case 1 implemented in *OMNET++*. The simulation is run for a hyperperiod which is 16 ms and the results are depicted in Table 7, where the observed and reported end-to-end (E2E) delays are shown in μs for *OMNET++* and *CACSS*, respectively.

Our validation experiment shows that the generated GCLs are correct and all the flows meet their requirements. The

values of the observed E2E delay from *OMNET++* (column 2) are equal to the values reported by *CACSS*-ZFD (column 3), which is expected, since they both use the same assumptions, e.g., ignoring the forwarding delay. Moreover, the maximum jitter is the same for all the solutions and equals to zero.

F. EVALUATION ON A HARDWARE PLATFORM

We have also evaluated the performance of *CACSS* on the hardware platform from [22] and in this context we removed the assumption that the forwarding delay is ignored. For this evaluation, we assumed that all the SWs are the same type as presented in [22]. The authors proposed the following equation for capturing the forwarding delay d in μs:

$$d = \lceil 4 + \frac{21 \times c}{400} \rceil, \tag{11}$$

where c is the size of the flows in bytes. Although we are using this TSN switch hardware implementation in a different application scenario compared to [22], since the forwarding delay model depends on the hardware implementation and not the application scenario, their delay model is also applicable to our case.

To be closer to implementation, *CACSS* can also consider the scheduling of PPTP flows for time synchronization. These PPTP flows have precedence constraint which has been already addressed in *CACSS*. We have considered that PPTP flows are implemented as high priority time sensitive traffic that are also scheduled along with network flows.

The generated GCLs are implemented on the SWs and the maximum delay and jitter of flows are measured from the measurements points shown in Fig. 8. We have used the three small "hardware test cases" from Table 5, where 4 flows are sent between ES1 and ES2 via SW1, 2 and 3. The three test cases differ in their flows' periods and deadlines, which are in the range of thousands of μs. The measurements were over several minuets using an oscilloscope resulting in hundreds of thousands of samples. The results are depicted in Table 5 where the columns 7 and 9 show the maximum delay and jitter values reported by *CACSS* and the columns 8 and 10 show the maximum delay and jitter values measured on the hardware platform. The deviation of the measured and reported maximum E2E delay values is small and is less than 1 μs for all the flows in all the test cases. Although, the measured maximum E2E jitter is non-zero for all the flows in all the test cases, the values are very small, in the nanoseconds range, without any effect on the deadlines or the control performance.

Let us illustrate the small variations measured in E2E delay for the hardware test case 2 from Table 5. Fig. 11 shows the measured E2E latencies in all samples for each flow, s_1 to s_4. The x-axis has the measured value of the E2E delay and the y-axis has the number of samples in which this value was measured. Although, as mentioned, the deviations are very small compared to the values reported by our *CACSS*, this shows the importance of considering realistic assumptions in the problem formulation. Note that the worst-case values of these variations can be added to the network precision δ introduced in Sect. V-B in order to guarantee that deadlines are satisfied.

VII. RELATED WORK

There is already a lot of research on Fog Computing, focused mostly on aspects related to quality-of-service (QoS) [35]–[37], with limited attention to safety-critical and real-time applications such as those used in the industrial areas. Real-time and safety-critical systems require guarantees for non-functional properties such as timing, e.g., that the deadlines are satisfied. Also, control applications have to fulfill non-functional properties related to control performance, e.g., QoC. Addressing the QoC for control applications in the Fog is still an open issue, researchers investigating the issue of degradation of control applications [38]–[40]. For example, [18] focuses on the routing and scheduling of messages of control applications to protect them from instability. The authors propose the control of the queue gates status via GCLs with careful consideration of the non-determinism of messages.

However, the area of co-design of control and real-time is a well studied area [41]–[46] which have tackled the design of controllers and scheduling of the control tasks and messages with respect to the control performance. The co-design procedure involves designing of control applications such that the controller is robust against degradation due to scheduling of the tasks and messages.

The control performance is not only affected by the scheduling of tasks but also affected by the scheduling of messages in networks. On one hand, researchers have addressed the configuration of communication aiming at increased control performance [41], [47], [48], but very few works address TSN. On the other hand, there is a lot of work on routing and scheduling for TSN, see the discussion below, but none of these works consider QoC. The work in [18] has considered routing and scheduling in Deterministic Ethernet, but lacks TSN-specific features which makes it difficult to implement the results, and uses an SMT formulation that cannot optimize the solutions and does not scale for large problem sizes. Our initial investigation in [16], [18] addresses QoC and considers the particularities of TSN, but uses a simplified model for control applications.

Researchers have addressed the routing and scheduling problems in TSN and have employed different approaches for the optimization, such as heuristics, metaheuristics and mathematical programming, e.g., ILP and Satisfiability Modulo Theories (SMT).

An example heuristic approach is [49], where the packets do not wait in switch queues, called no-wait scheduling. The authors propose a Tabu Search metaheuristic to optimize the flowspan which may become larger because of the no-wait scheduling, and also let lower-priority traffic to use the residual bandwidth. Wisniewski et. al in [50] increase the flexibility of the scheduling by employing a greedy-based

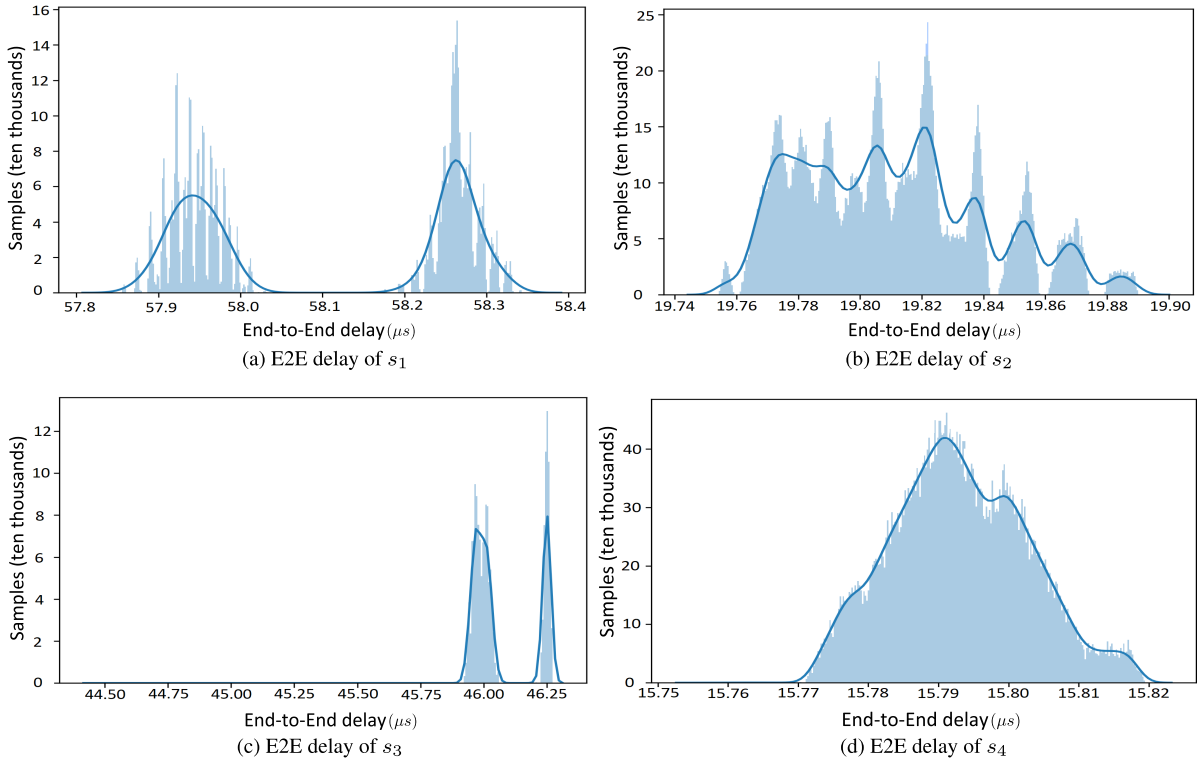

FIGURE 11. The details of the measured E2E delay of flows in the test case 2 form the Table 5 implemented on the hardware platform. Thick lines are kernel density estimates.

heuristic approach which is less resource demanding, and possible to be implemented on industrial equipment on the field floor. The greedy-based heuristic approach is also proposed in [51], where authors aim to generate joint network routing and communication scheduling that are fault-tolerant, within a reasonably short time. Arestova *et al.* in [52] propose a hybrid genetic algorithm for the communication scheduling and network routing to find a near-optimal solution in a reasonable time, and also optimizing the bandwidth to let more less-critical traffic transmitted. A heuristic list scheduler for joint communication scheduling and network routing is proposed in [53], where multi-cast traffic and application distribution are allowed, and bandwidth is optimized. The same problem is addressed in [54] where a genetic algorithm is employed and in [23], where multiple traffic types are considered.

The use of SMT solvers for the communication scheduling is first proposed in [55]. The author proposes a general method for off-line scheduling of communication and uses the SMT solver as the back-end solver. The SMT-based model for TT-schedules shows promising results and scales well with the problem size. Craciunas *et al.* in [20] propose an SMT model for the traffic scheduling which generates solutions that are jitter-free and the number of used port queues in the network switches is minimized. The authors also propose frame and flow isolation constraints and evaluate them on several tests concerning the run-time and number of used queues. Craciunas *et al.* derive general traffic regulating constraints for SMT solvers in [56], which introduces windows in GCLs and maps the frames to them. Another SMT model based on ''array theory encoding'' is proposed in [33], where the authors see the GCL windows as array elements, letting more relaxed scheduling with allowing jitter and having fewer GCL entries. However, the implementation of the proposed method shows resource demanding. The trade-of between the GCL length and run-time is well studied in [57].

The SMT-based schedulers are extended for the benefit of other applications. For example, in [58], authors combine traffic scheduling and network routing problem to achieve the minimum delay for AVB traffic. The traffic scheduling combined with task scheduling is studied in [24], where an SMT solver is employed to schedule network messages and tasks on a networked computation platform which is equipped with time-triggered network. Park *et al.* in [59] proposes a generic algorithm approach to schedule the communication in TSN where preemption is allowed. The proposed algorithm shows increased reliability in the generated solutions. The communication scheduling concerning the security of control applications is addressed in [60] where the authors aim to increase the resilience of the control applications to malicious interference.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we have addressed the problem of scheduling real-time traffic via TSN on an FCP, aiming at improving the performance of industrial control applications and addressing the timing requirements of real-time applications. The scheduled traffic in TSN is regulated through the Gate Control

Lists (GCLs), which allow the transmission of flows by opening and closing the switch gates.

We have proposed a Constraint Programming-based solution for determining the GCLs such that the control performance (in terms of QoC) is maximized and the deadlines are satisfied. The solution models the problem through a set of constraints and uses an QoC analytical model inside the objective function for optimizing the solution. We also employ a metaheuristic search strategy to drive the search faster towards good quality solutions in a short time. Our CP solution for messages is extensible and can be integrated with CP task scheduling models from the literature. In addition, we aimed at introducing space in the timeline of message schedules, increasing thus the probability of successfully integrating our GCLs with the tasks running on the end systems.

As the results show, the solution has successfully scheduled the flows in all test cases and also achieved a good QoC for control applications. We have used OMNET++ and JitterTime for validating the results and the performance of the QoC analytical model proposed. We have also implemented the resulted GCLs on a TSN hardware platform.

REFERENCES

[1] T. J. Williams, "The Purdue enterprise reference architecture," *Comput. Ind.*, vol. 24, nos. 2–3, pp. 141–158, Sep. 1994.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.

[3] P. Mell and T. Grance, "The NIST definition of cloud computing," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. SP 800-145, 2011.

[4] *IEEE Standard for Adoption of Openfog Reference Architecture for Fog Computing*, IEEE Standard 1934-2018, 2018, pp. 1–176.

[5] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for Internet of Things and analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments*. Cham, Switzerland: Springer, 2014, pp. 169–186.

[6] IEEE. (2016). *Official Website of the 802.1 Time-Sensitive Networking Task Group*. Accessed: Dec. 26, 2020. [Online]. Available: http://www.ieee802.org/1/pages/tsn.html

[7] E. Dahlman, G. Mildh, S. Parkvall, J. Peisa, J. Sachs, Y. Selén, and J. Sköld, "5G wireless access: Requirements and realization," *IEEE Commun. Mag.*, vol. 52, no. 12, pp. 42–47, Dec. 2014.

[8] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*. Berlin, Germany: Springer, 2009.

[9] C. Puliafito, E. Mingozzi, F. Longo, A. Puliafito, and O. Rana, "Fog computing for the Internet of Things: A survey," *ACM Trans. Internet Technol.*, vol. 19, no. 2, pp. 1–41, Apr. 2019.

[10] P. Hu, S. Dhelim, H. Ning, and T. Qiu, "Survey on fog computing: Architecture, key technologies, applications and open issues," *J. Netw. Comput. Appl.*, vol. 98, pp. 27–42, Nov. 2017.

[11] P. Pop, B. Zarrin, M. Barzegaran, S. Schulte, S. Punnekkat, J. Ruh, and W. Steiner, "The FORA fog computing platform for industrial IoT," *Inf. Syst.*, vol. 98, May 2021, Art. no. 101727.

[12] TTTech Computertechnik AG. (2019). *Nerve*. Accessed: Dec. 26, 2020. [Online]. Available: https://www.tttech-industrial.com/products/nerve

[13] P. Gaj, J. Jasperneite, and M. Felser, "Computer communication within industrial distributed environment—A survey," *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 182–189, Feb. 2013.

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

[15] *Timing and Synchronization for Time-Sensitive Applications*, IEEE Standard 802.1ASRev, 2017. [Online]. Available: http://www.ieee802.org/1/pages/802.1AS-rev.html

[16] M. Barzegaran, B. Zarrin, and P. Pop, "Quality-of-control-aware scheduling of communication in TSN-based fog computing platforms using constraint programming," in *Proc. 2nd Workshop Fog Comput. IoT*, vol. 80, 2020, pp. 3:1–3:9.

[17] M. Barzegaran, A. Cervin, and P. Pop, "Performance optimization of control applications on fog computing platforms using scheduling and isolation," *IEEE Access*, vol. 8, pp. 104085–104098, 2020.

[18] R. Mahfouzi, A. Aminifar, S. Samii, A. Rezine, P. Eles, and Z. Peng, "Stability-aware integrated routing and scheduling for control applications in Ethernet networks," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 682–687.

[19] *Bridges and Bridged Networks*, Standard 802.1Q-2014, Institute of Electrical and Electronics Engineers, 2014. [Online]. Available: http://www.ieee802.org/1/pages/802.1Q.html

[20] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in IEEE 802.1Qbv time sensitive networks," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, 2016, pp. 183–192.

[21] A. Cervin, "Integrated control and real-time scheduling," Ph.D. dissertation, Dept. Autom. Control, Lund Univ., Lund, Sweden, 2003.

[22] J. Sanchez-Garrido, A. Jurado, L. Medina, R. Rodriguez, E. Ros, and J. Diaz, "Digital electrical substation communications based on deterministic time-sensitive networking over Ethernet," *IEEE Access*, vol. 8, pp. 93621–93634, 2020.

[23] V. Gavrilut, L. Zhao, M. L. Raagaard, and P. Pop, "AVB-aware routing and scheduling of time-triggered traffic for TSN," *IEEE Access*, vol. 6, pp. 75229–75243, 2018.

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

[25] K. Ogata and Y. Yang, *Modern Control Engineering*, vol. 4. Delhi, India: Prentice-Hall, 2002.

[26] K. Astrom and B. Wittenmark, *Computer Controlled System*. Upper Saddle River, NJ, USA: Prentice-Hall, 1997.

[27] A. Cervin, P. Pazzaglia, M. Barzegaran, and R. Mahfouzi, "Using JitterTime to analyze transient performance in adaptive and reconfigurable control systems," in *Proc. 24th IEEE Int. Conf. Emerg. Technol. Factory Automat. (ETFA)*, Sep. 2019, pp. 1025–1032.

[28] B. Lincoln and A. Cervin, "JITTERBUG: A tool for analysis of real-time control performance," in *Proc. 41st IEEE Conf. Decis. Control*, vol. 2, Dec. 2002, pp. 1319–1324.

[29] D. Henriksson, A. Cervin, and K.-E. Årzén, "TrueTime: Simulation of control loops under shared computer resources," *IFAC Proc. Volumes*, vol. 35, no. 1, pp. 417–422, 2002.

[30] O. Sinnen, *Task Scheduling for Parallel Systems*, vol. 60. Hoboken, NJ, USA: Wiley, 2007.

[31] Google. *Google OR-Tools*. Accessed: Dec. 26, 2020. [Online]. Available: https://developers.google.com/optimization

[32] E. K. Burke and G. Kendall, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, 2nd ed. New York, NY, USA: Springer, 2013.

[33] R. S. Oliver, S. S. Craciunas, and W. Steiner, "IEEE 802.1Qbv gate control list synthesis using array theory encoding," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2018, pp. 13–24.

[34] J. Falk, D. Hellmanns, B. Carabelli, N. Nayak, F. Dürr, S. Kehrer, and K. Rothermel, "NeSTiNg: Simulating IEEE time-sensitive networking (TSN) in OMNeT++," in *Proc. Int. Conf. Netw. Syst. (NetSys)*, Mar. 2019, pp. 1–8.

[35] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the Internet of Things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[36] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: A taxonomy, survey and future directions," in *Internet of Everything*. Singapore: Springer, 2018, pp. 103–130.

[37] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 416–464, 1st Quart., 2018.

[38] X.-M. Zhang, Q.-L. Han, and X. Yu, "Survey on recent advances in networked control systems," *IEEE Trans. Ind. Informat.*, vol. 12, no. 5, pp. 1740–1752, Oct. 2016.

[39] D. Zhang, P. Shi, Q.-G. Wang, and L. Yu, "Analysis and synthesis of networked control systems: A survey of recent advances and challenges," *ISA Trans.*, vol. 66, pp. 376–392, Jan. 2017.
[40] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu, "A survey of recent results in networked control systems," *Proc. IEEE*, vol. 95, no. 1, pp. 138–162, Jan. 2007.
[41] Z. Huo and Z. Zhang, "Scheduling and control co-design for networked wind energy conversion systems," *Global Energy Interconnection*, vol. 2, no. 4, pp. 328–335, Aug. 2019.
[42] F. Smarra, A. D'Innocenzo, and M. D. Di Benedetto, "Optimal co-design of control, scheduling and routing in multi-hop control networks," in *Proc. IEEE 51st IEEE Conf. Decis. Control (CDC)*, Dec. 2012, pp. 1960–1965.
[43] P. Martí, J. Yépez, M. Velasco, R. Villà, and J. M. Fuertes, "Managing quality-of-control in network-based control systems by controller and message scheduling co-design," *IEEE Trans. Ind. Electron.*, vol. 51, no. 6, pp. 1159–1167, Dec. 2004.
[44] K.-J. Park, M.-K. Yoon, K. Kang, and C.-G. Lee, "Scheduling and control co-design under end-to-end response time constraints in cyber-physical systems," in *Proc. IEEE Conf. Comput. Commun. Workshops*, Apr. 2011, pp. 762–767.
[45] M. S. Branicky, S. M. Phillips, and W. Zhang, "Scheduling and feedback co-design for networked control systems," in *Proc. 41st IEEE Conf. Decis. Control*, vol. 2, Dec. 2002, pp. 1211–1217.
[46] D. Simon, A. Seuret, and O. Sename, "Real-time control systems: Feedback, scheduling and robustness," *Int. J. Syst. Sci.*, vol. 48, no. 11, pp. 2368–2378, Aug. 2017.
[47] Z.-W. Wang and H.-T. Sun, "Control and scheduling co-design of networked control system: Overview and directions," in *Proc. Int. Conf. Mach. Learn. Cybern.*, vol. 3, Jul. 2012, pp. 816–824.
[48] Y.-Q. Song, "Networked control systems: From independent designs of the network QoS and the control to the co-design," *IFAC Proc. Volumes*, vol. 42, no. 3, pp. 155–162, 2009.
[49] F. Dürr and N. G. Nayak, "No-wait packet scheduling for IEEE time-sensitive networks (TSN)," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, 2016, pp. 203–212.
[50] L. Wisniewski, M. Schumacher, J. Jasperneite, and C. Diedrich, "Increasing flexibility of time triggered Ethernet based systems by optimal greedy scheduling approach," in *Proc. IEEE 20th Conf. Emerg. Technol. Factory Automat. (ETFA)*, Sep. 2015, pp. 1–6.
[51] A. A. Atallah, G. B. Hamad, and O. A. Mohamed, "Fault-resilient topology planning and traffic configuration for IEEE 802.1Qbv TSN networks," in *Proc. IEEE 24th Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2018, pp. 151–156.
[52] A. Arestova, K.-S. J. Hielscher, and R. German, "Design of a hybrid genetic algorithm for time-sensitive networking," in *Measurement, Modelling and Evaluation of Computing Systems*. Cham, Switzerland: Springer, 2020.
[53] M. Pahlevan, N. Tabassam, and R. Obermaisser, "Heuristic list scheduler for time triggered traffic in time sensitive networks," *SIGBED Rev.*, vol. 16, no. 1, pp. 15–20, Feb. 2019.
[54] M. Pahlevan and R. Obermaisser, "Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks," in *Proc. IEEE 23rd Int. Conf. Emerg. Technol. Factory Automat. (ETFA)*, vol. 1, Sep. 2018, pp. 337–344.
[55] W. Steiner, "An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks," in *Proc. IEEE Real-Time Syst. Symp.*, Nov. 2010, pp. 375–384.
[56] S. S. Craciunas, R. S. Oliver, and W. Steiner, "Formal scheduling constraints for time-sensitive networks," 2017, *arXiv:1712.02246*. [Online]. Available: http://arxiv.org/abs/1712.02246
[57] W. Steiner, S. S. Craciunas, and R. S. Oliver, "Traffic planning for time-sensitive communication," *IEEE Commun. Standards Mag.*, vol. 2, no. 2, pp. 42–47, Jun. 2018.
[58] P. Pop, M. L. Raagaard, S. S. Craciunas, and W. Steiner, "Design optimisation of cyber-physical distributed systems using IEEE time-sensitive networks," *IET Cyber-Phys. Syst., Theory Appl.*, vol. 1, no. 1, pp. 86–94, Dec. 2016.
[59] T. Park, S. Samii, and K. G. Shin, "Design optimization of frame preemption in real-time switched Ethernet," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 420–425.
[60] R. Mahfouzi, A. Aminifar, S. Samii, P. Eles, and Z. Peng, "Security-aware routing and scheduling for control applications on Ethernet TSN networks," *ACM Trans. Design Automat. Electron. Syst.*, vol. 25, no. 1, pp. 1–26, Jan. 2020.

MOHAMMADREZA BARZEGARAN (Graduate Student Member, IEEE) has been a Marie Curie Ph.D. Fellow in computer science with the Technical University of Denmark, since 2018. His research is focused on the configuration of Fog computing platform for critical control applications. His main research interests include optimization, configuration of real-time and safety-critical systems, and co-design of control applications for real-time and safety-critical systems.

PAUL POP (Member, IEEE) received the Ph.D. degree in computer systems from Linköping University, in 2003. He has been an Associate Professor with DTU Compute, Technical University of Denmark, and since 2016, he has been a Professor of Cyber-Physical Systems. His research interest includes developing methods and tools for the analysis and optimization of dependable embedded systems. In this area, he has published over 130 peer-reviewed articles, three books, and seven book chapters. His research has been highlighted as "The Most Influential Papers of 10 Years DATE". He has served as a Technical Program Committee Member for several conferences, such as DATE and ESWEEK. He has received the Best Paper Award from DATE 2005, RTIS 2007, CASES 2009, MECO 2013, and DSD 2016. He has also received the EDAA Outstanding Dissertations Award (Co-Supervisor) in 2011. He is the Chairman of the IEEE Danish Chapter on Embedded Systems and the Director of the DTU's IoT Research Center. He has coordinated the Danish National InfinIT Safety-Critical Systems Interest Group.