Configuration Optimization of Fog Computing Platforms for Control Applications

Mohammadreza Barzegaran

Kongens Lyngby 2021

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary (English)

Industry 4.0 requires the convergence of Operational and Information Technologies (OT & IT), which use different computation and communication technologies. Cloud Computing cannot be used for OT involving industrial applications since it cannot guarantee stringent non-functional requirements, e.g., dependability, trustworthiness and timeliness. Instead, a new computing paradigm, called Fog Computing, is envisioned as an architectural means to realize the IT/OT convergence. A Fog Computing Platform (FCP) brings computing and deterministic communication closer to the network's edge, where the machines are located in industrial applications. An FCP is implemented as a set of Fog Nodes (FNs) that integrate communication, computation, and storage resources. 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.

With the IT/OT convergence, applications of mixed-criticality will share the same FCP. At one extreme, we have the safety-critical real-time systems that control industrial process and have to be operational even in the case of failure. The vision is to virtualize these as applications composed of tasks and messages running on an FCP. At the other extreme, we have non-critical Fog applications that do not have stringent timing and dependability requirements but are required to implement the novel functionalities of Industry 4.0.

We assume that the platform uses partitioning to enforce the spatial and temporal isolation between applications with different criticalities. Applications are modeled as tasks interacting via messages transmitted as flows on TSN. We consider several scheduling policies for tasks within a hierarchical scheduling model that can address the varied time-criticality requirements of applications. For example, the critical control applications are scheduled using static cyclic scheduling, and the resources of the Fog applications are allocated at runtime using best effort policies.

We propose several approaches to the design time FCP configuration optimization for mixed-criticality applications, such that the performance (in terms of Quality-of-Control) and timeliness of control applications are guaranteed, and the Quality-of-Service of non-critical Fog applications is maximized. In addition, we are interested in extensible configurations that support the addition of future new control applications and can successfully accommodate at runtime a large number of responsive Fog applications. At runtime, our approaches handle the migration and best–effort scheduling of Fog applications to the FNs that have resources for their execution. Determining an FCP configuration means: deciding the partitions that provide temporal and spatial isolation among mixed-criticality applications, mapping the tasks to the cores of the multicore FNs, routing of flows on TSN, synthesizing the task schedule tables and the Gate Control Lists for switches that schedule the transmission of flows.

We have developed several algorithms that use heuristics, metaheuristics and Constraint Programming to solve these combinatorial optimization problems. The algorithms have been extensively evaluated on several test cases, including realistic test cases from the industry.

Summary (Danish)

Industri 4.0 forudsætter konvergens mellem operationelle teknologier og informationsteknologier (OT& IT), der gør brug af forskellige beregnings- og kommunikationsteknologier. Cloud Computing kan ikke bruges til OT, der involverer industrielle applikationer, da det ikke kan garantere strenge ikke-funktionelle krav, f.eks. pålidelighed, pålidelighed og aktualitet. I stedet foreslås forestilles et nyt databehandlingsparadigme, kaldet Fog Computing, som et arkitektonisk middel til at realisere IT/OT-konvergensen. En Fog Computing Platform (FCP) bringer beregninger og deterministisk kommunikation tættere på netværkets kant, hvor maskinerne er placeret i industrielle applikationer. En FCP implementeres som et sæt Fog Noder (FN'er), der integrerer kommunikations-, beregnings- og lagerressourcer. I lighed med tidligere forskning og igangværende standardiseringsarbejde antager vi, at kommunikationen mellem FN'er opnås via IEEE 802.1 Time-Sensitive Networking standarden (TSN).

Med IT/OT-konvergensen vil applikationer med mixed-criticality dele den samme FCP. På den ene yderste side har vi de sikkerhedskritiske realtidssystemer, der styrer den industrielle proces og skal være operationelle, selv i tilfælde af fiasko. Visionen er at virtualisere disse som applikationer sammensat af opgaver og meddelelser, der kører på en FCP. På den anden side har vi de ikke-kritiske Fog-applikationer, der ikke har strenge krav til timing og pålidelighed, men som er nødvendige for at implementere de nye funktioner i Industri 4.0.

Vi antager, at platformen udnytter opdeling til at opfylde den rummelige og tidsmæssige isolering mellem applikationer med forskellige kritiske egenskaber. Applikationer modelleres som opgaver, der interagerer via meddelelser, der transmitteres som strømme på TSN. Vi overvejer flere planlægningsprincipper for opgaver inden for en hierarkisk planlægningsmodel, der kan opfylde de forskellige tidskritiske krav til forskellige

anvendelser. Eksempelvis planlægges de kritiske kontrolapplikationer ved anvendelse af statisk cyklisk planlægning, mens ressourcerne i Fog-applikationerne tideles ved kørsel baseret på bedste indsats procedurer.

Vi foreslår adskillige tilgange til designtid FCP-konfigurationsoptimering til applikationer med mixed-criticality, således at ydeevnen (med hensyn til kvalitetskontrol) og aktualiteten af kontrolapplikationer er garanteret, og servicekvaliteten for ikke-kritiske Fog-applikationer er maksimeret. Derudover er vi interesseret i skalerbare konfigurationer, der understøtter tilføjelsen af nye fremtidige kontrolapplikationer, og som undereksekvering kan sikre et stort antal responsive Fog-applikationer under kørsel. Ved eksekvering sikrer vores system migrering og planlægning af optimal indsats for Fog-applikationer til FN'er, der har ledige ressourcer til deres udførelse. Bestemmelse af en FCP-konfiguration betyder: at beslutte de opdelinger, der giver tidsmæssig og rummelig isolering blandt applikationer med blandet kritisk betydning, kortlægge opgaverne til kernerne i multicore FN'erne, routing af strømme på TSN, syntetisering af opgavetabellen og Gate Control Lists for switcher, der planlægger transmission af strømme.

Vi har udviklet flere algoritmer, der bruger heuristik, metaheuristik og begrænsningsprogrammering til at løse disse kombinatoriske optimeringsproblemer. Algoritmerne er blevet udførligt evalueret i flere testeksempler, herunder realistiske testeksempler fra branchen.

Preface

This thesis was prepared at Department of Applied Mathematics and Computer Science (DTU Compute) to fulfill the requirements for acquiring a Ph.D. degree in Computer Science.

In this thesis, I propose analysis and optimization methods for configuring Fog Computing Platforms that host mixed-criticality applications, including critical control applications, aiming to guarantee their performance and timeliness. The thesis consists of an introductory chapter and four papers.

The work has been supervised by Professor Paul Pop and co-supervised by Professor Jan Madsen.

Lyngby, 14-May-2021
Mohammadreza Barzegaran

Barzegaran

Acknowledgments

First of all, I do not know how to thank and show appreciation to my supervisor, Prof. Paul Pop. During the course of my Ph.D. and despite my different academic background, his guidance helped me in all aspects of my research endeavor. He also gave me the opportunity and encouragement to strengthen my profile as a future high-skilled researcher. I am deeply thankful for his constant, broad, professional, and constructive feedback, which helped me fill the gaps in my training.

I notably appreciate Dr. Bahram Zarrin, who I had the opportunity to work with in the FORA ETN— European Training Network on Fog Computing for Robotics and Industrial Automation, for his support and guidance. I want to thank Assoc. Prof. Stefan Schulte (from TU Wien, Austria), for his hospitality and advice during my external stay in FORA. My special appreciation goes towards Dr.Juha Kuusela for the insightful conversations we had during my FORA external stay at Danfoss Drives A/S.

I would like to express a big thank you to Assoc. Prof. Anton Cervin (from Lund University, Sweden) for his guidance and constructive discussions related to control and real-time co-design, which helped me early in my Ph.D. studies. I highly appreciate the support of the administrative and technical staff, especially Karin Tunder and Ellen Juel Nielsen, the ones who always helped me promptly. I would also like to thank all fellow students and supervisors in FORA project whom we had a great time together.

Finally, I would like to thank my family for their love and support from the deep down of my heart. My special gratitude goes to my sister, Dr. Marieh Barzegaran, for her valuable virtual presence in my life from thousands of miles away. I take the opportunity to say “I love you” to my parents, Parivash and Naser. I also say a huge thank you to Aliyeh, Naser, Mahsa, Peyman, Mohammadreza, Farnaz and little Ava who support me and became my new family.

Last but not least, I acknowledge that the research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764785, FORA—Fog Computing for Robotics and Industrial Automation.

Abbreviations

AADL	Architecture Analysis & Design Language
AC	Alternating Current
AI	Artificial Intelligence
AVB	Audio-Video Bridging
BE	Best Effort
CACSS	Control-Aware Communication Scheduling Strategy
CP	Constraint Programming
CPS	Cyber-Physical System
DAG	Directed Acyclic Graph
DC	Direct Current
ECOS	Extensible Configuration Optimization Strategy
EDF	Earliest Deadline First
ES	End-System
FCP	Fog Computing Platform
FCS	Feedback Control System
FIFO	First-In-First-Out
FN	Fog Node
GCL	Gate Control List
HMI	Human–Machine Interface
IIoT	Industrial Internet of Things
ILP	Integer-Linear Programming

IoT	Internet of Things
IPC	Industrial Personal Computer
IT	Information Technology
KPI	Key Performance Indicator
MILP	Mixed Integer-Linear Programming
ML	Machine Learning
MTU	Maximum Transmission Unit
NRE	Non-Recurring Engineering
OPC UA	OPC Unified Architecture
OT	Operational Technology
PCP	Priority Code Point
PLC	Programmable Logic Controller
PPTP	Point-to-Point Tunneling Protocol
QoC	Quality of Control
QoS	quality-of-service
RM	Rate Monotonic
SA	Simulated Annealing
SAE	Society of Automotive Engineers
SIL	Safety Integrity Level
SMT	Satisfiability Modulo Theories
ST	Scheduled Traffic
SW	Switch
TSN	Time-Sensitive Networking
TT	Time-Triggered
UC	Use Case
WCET	Worst-Case Execution Time

List of Figures

List of Tables

Contents

CHAPTER 1

Introduction

1.1 Motivation and Background

The third industrial revolution (“Industry 3.0”) began around 1970 with the first computer era and transformed mechanical and analog electronic technologies into digital electronics in the form of digital computing and communication technologies [ZLZ15]. Industry 3.0 involved using these technologies to run control systems for automation in production and other industrial systems, which brought innovations and new business opportunities. The control systems and resulting automation advantage productivity, safety, and quality. An example benefit in manufacturing is the production of large amounts of standardized products in a constant flow, i.e., mass production [Bea04].

A *control system* operates and commands a dynamical system (for example, robots and industrial machines) using a control algorithm [OY02]. The essential ingredients of a control system are: (i) objectives of control that are inputs of the control systems, (ii) control system components that are computers and computer-based algorithms, and (iii) results or outputs that are the control signals applied to physical entities [GK17]. To this end, control systems use sensors to sample the dynamical system that show the system’s current state and calculate the deviation from a desired set-point. The control algorithm calculates output signals based on the deviation from the desired set-point. It drives the dynamical system to the desired set-point by applying the control signal via actuators. An example of a control system is the pressure regulator in a factory that maintains the fluid pressure. In this example, the pressure regulator uses a sensor to measure the fluid's current pressure as the input. It calculates the deviation of the current fluid pressure from a desired pressure value, determines the appropriate valve opening value to maintain the desired fluid pressure, and applies the determined valve opening value via the actuator, which is the valve positioner.

These sampling and actuation impose real-time constraints on the control system. As defined by Kopetz, "A real-time computer system is a computer system where the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical time when these results are produced" [Kop11]. A common misconception is that "A real-time computer system is able to quickly produce the results of computations" [But11], however, real-time computing is about predictability—even during the worst-case behavior, not about speed [Kop11, But11].

A real-time system can be classified from the perspective of its timing constraints, e.g., deadlines. A *deadline* is a time instant when the results of the real-time computation must be produced, and it is imposed on the activities of a real-time system, depending on the characteristics of the implemented application [Kop11]. On the one hand, a hard real-time system requires results of the computation to be produced before or on the deadline, called a *hard deadline*, otherwise its operation is incorrect. On the other hand, a soft real-time system has less stringent timing constraints, and missing a *soft deadline* does not invalidate the correctness of the results but leads to performance degradation [Kop11]. Furthermore, missing a deadline in a hard real-time system is considered as a failure. However, missing a deadline in a soft real-time system instead degrades the usefulness of the results of computation resulting in a lower quality-of-service [SR94].

Real-time systems are often also *safety-critical systems* [Kni02], whose failure may result in death or serious injury to people, loss or severe damage to equipment/property, or environmental harm [Kni02]. Aircraft flight control and medical devices are examples of such safety-critical systems, where a failure may cause injury or death to people. This criticality of safety functions implemented by safety-critical real-time systems is determined via risk assessment and failure analysis methods [Rau14].

A *mixed-criticality system* is defined as "an integrated suite of hardware, operating system, middleware services, and application software that supports the execution of safety-critical, mission-critical, and non-critical software within a single, secure computing platform" [BBB$^+$09]. A failure in a mission-critical system will result in a serious impact on its normal operation. However, a non-critical system being inessential can become operational again after a failure.

Industry 3.0 uses Operational Technology (OT), which relies on dedicated hardware and software that implement the control systems and process the control data with real-time requirements [Glo21]. OT provides guarantees for real-time requirements and has a high degree of dependability. Examples of technologies used in OT are

Industrial Personal Computers (IPCs), which are computers that have been ruggedized and configured for industrial applications, Programmable Logic Controllers (PLCs), which are computers that run real-time operations and control machines, and time-optimized safety-critical proprietary communication protocols.

The real-time requirements of industrial applications have so far been fulfilled via OT systems that are statically configured and use overprovisioning, with no support for dynamic changes and reconfigurations [Eur16]. OT is not suited for business intelligence applications or Big Data and analytics due to technological constraints such as limited communication bandwidth and limited computation resources [Eur16, HGB14]. Additionally, OT systems are often expensive due to the absence of open and standards-based solutions, the lock-in by specific vendors and the confines of their product development plans [Eur16].

On the contrary, Information Technology (IT) uses different computation and communication technologies that are optimized for dealing with increased scalability and performance, storing and manipulating data [RL19]. IT brings flexibility and capabilities for faster development and improvement with Cloud Computing, Artificial Intelligence (AI), and Big Data. However, IT is not directly applicable to industrial applications where non-functional properties such as timeliness and dependability have to be guaranteed [GVCL14].

1.2 Fog Computing

The term “IT/OT convergence” refers to the merging of technologies from IT systems with OT systems, which are using separated computation and communication paradigms [Eur16, KHS10]. The evolution of novel technological paradigms via the IT/OT convergence will bring effectiveness, flexibility, connectivity, interoperability, scalability, and capabilities for faster development and improvement with Cloud Computing, AI, and Big Data in industrial systems [GLSC17]. Although the IT/OT convergence has many challenges due to vendor lock-in and the absence of open and standard-based solutions [Eur16]), it is required in order to enable Industry 4.0 [Gil16]. Thus, both industry and academia have made a significant effort to promote the convergence of IT/OT [PML$^+$19, HDNQ17].

The term “Fourth Industrial Revolution” (also called *Industry 4.0*) was first introduced by a team of scientists developing a high-tech strategy for the German government [KLW11]. The term was promoted to a wider audience by Klaus Schwab [Sch17] who includes it in technologies that combine hardware, software, and biology, with an emphasis on advances in communication and connectivity [Sch21]. Schwab expects this new industrial revolution to be marked by breakthroughs in emerging technologies

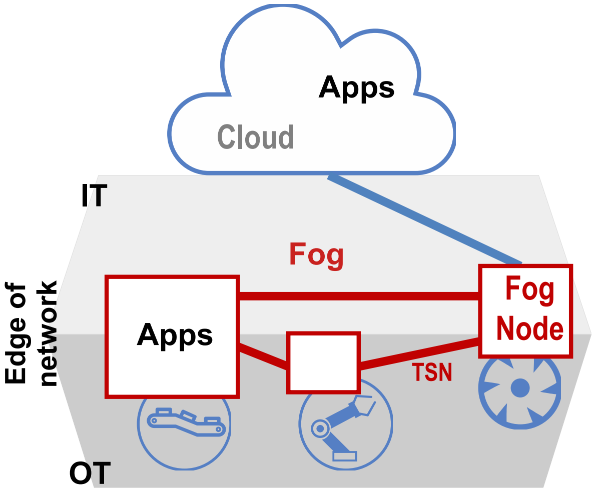


Figure 1.1: A Fog Computing Platform consists of Fog Nodes (boxes) running applications (Apps) and are connected to the Cloud and to each other via physical links (thick lines) [BP21a].

in fields such as robotics, artificial intelligence, nanotechnology, quantum computing, biotechnology, the internet of things, the Industrial Internet of Things (IIoT), decentralized consensus, fifth-generation (5G) wireless technologies, 3D printing, and fully autonomous vehicles [Sch21].

Industry 4.0 will be realized via interconnected *Cyber-Physical Systems (CPSs)* that integrate computation with the physical process [Lee08] and create the capabilities required for IIoT [Erb17]. Moreover, Industry 4.0 requires the IT/OT convergence as its enabler [Gil16]. Fog Computing, is envisioned as an architectural means to realize the vision of Industry 4.0. *Fog Computing* is a “system-level architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Things” [Fog21]. Fog Computing is distinguished from *Edge Computing*, which is defined as an architecture in which the resources of an edge server are placed at the edge of the Internet, in close proximity to CPSs, mobile devices, sensors, and IIoT endpoints.

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 efficiency and flexibility [BMNZ14, YLL15, BMZA12, MKB18]. The vision of Fog Computing is to provide the same dependability level of OT but instead virtualize control systems as software running in the Fog.

A Fog Computing Platform (FCP) consists of Fog Nodes (FNs) that integrate communication and computation resources to enable a variety of communication and computation options (see Fig. 1.1). FNs can have different computation and communication capabilities and can be used together in the same FCP to host applications with various resource demands [PRGS18]. Several solutions for implementing FNs have been proposed [PML$^+$19, HDNQ17, BMNZ14] and several FN solutions have been developed by companies [TTT21, Neb21].

With a deterministic communication solution integrated into an FCP, the FN (that virtualizes the control system) can be placed at a further spatial distance from the dynamical system (that the FN controls). This vision is realized via interconnected CPSs, virtualization solutions [LJYZ17], standardized Deterministic Ethernet solutions from IEEE Time-Sensitive Networking (TSN) Task Group [IEE21b], upcoming 5G wireless standards [DMP$^+$14], and interoperability standards such as OPC Unified Architecture (OPC UA) [MLD09].

1.2.1 The FORA Fog Computing Platform

The European Training Network on Fog Computing for Robotics and Industrial Automation (FORA ETN) [PZB$^+$21] is one of the initiatives working on the realization of an FCP. FORA has proposed an FCP reference architecture targeting IIoT applications [PZB$^+$21]. The FORA FCP is focused on the virtualization of industrial control, which is implemented as control applications.

The FORA FCP reference architecture is defined using the Architecture Analysis & Design Language (AADL), which is a well-known architecture description language in the domain of real-time embedded systems [FGH06]. The AADL model captures the main components and their interconnections in the FCP [PZB$^+$21] and discusses their possible implementation using “Technology Bricks”, which can be a hardware or software prototype, a method, a tool, a model, etc.

The FNs of the FORA FCP contain multicore CPUs, memory, I/Os, communication cards for TSN, Ethernet, wireless in terms of hardware, and the software stack includes, hypervisors, middleware, and different policies and mechanisms for resource management, see the following sections for more information. Fig. 1.2 shows an overview of the FORA FN design using AADL.

Furthermore, the FORA FCP is equipped with deterministic networking which is realized using open standards such as IEEE 802.1 TSN [IEE21b] and relies on interoperability standards such as OPC UA [MLD09]. A TSN network guarantees bounded latency communication between FNs of an FCP and its environment. This guarantee enables the relocation of real-time critical applications from machines to FNs [PZB$^+$21].

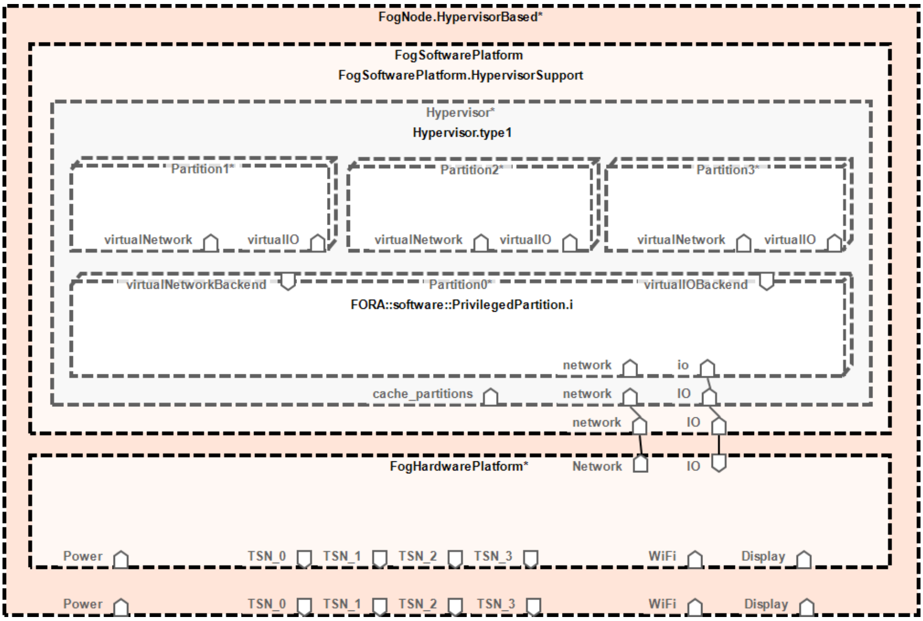


Figure 1.2: Overview of the FORA Fog node design [PZB+21]

A TSN switch can be part of an FN, which is the case in some areas such as automotive, or a separate device [PZB+21], which is typical, for example, in industrial automation.

The FORA FCP is capable of hosting applications that are assigned to run on FNs, e.g., critical control applications that are configured at design-time and dynamic Fog applications that may be migrating in and out of FNs and have to be handled at runtime. To this end, the FORA FCP proposes using deterministic virtualization that helps the timeliness and reduces the effort required for safety and security assurance. The deterministic virtualization is realized by integrating hypervisors into FNs, see Sect. 1.2.1.2. Additionally, the FORA FCP uses middleware for supporting both critical control and dynamic Fog applications [PZB+21]. The FORA Fog middleware will also build on application layer protocols such as MQTT-SN [SCT13] for northbound communication and TSN for southbound communication.

Throughout this thesis, we assume that the Industry 4.0 mixed-criticality applications are implemented using the FORA FCP. These applications are protected from each other using the technologies presented in the following sections.

1.2.1.1 Resource Management

An FCP can contain various types of FNs, from low-end to high-end concerning their communication and computation capabilities [PRGS18]. However, this different computational and communication resources in the FCP comes with challenges in deploying and managing the applications on the FNs which have different requirements and demands [GASR19]. Thus, the FCP needs to monitor and manage its resources, similar to the Cloud Computing that automatically manages its resources to scale and handle increasing workload [MNH$^+$15].

There are various resource management approaches in the literature targeting different metrics [BM20, HV19]. Most of the resource management approaches in the literature aim to provide low-latency and energy efficiency [BM20], and cannot provide real-time and dependability guarantees.

The goal of resource management and configuration techniques in FORA FCP reference architecture is to provide the necessary computation and communication resources to all applications via FNs, balance the overall resource utilization landscape, and provide real-time guarantees for critical applications [PZB$^+$21].

1.2.1.2 Hypervisors

Virtualization is the act of creating a virtual version of computer hardware platforms, storage devices, and computer network resources [Wik21b]. It is realized via *hypervisors* that are computer software or firmware [IBM21]. A hypervisor creates an abstraction layer over the resources of a “host machine”, allowing to divide the resources between different applications, e.g., mixed-criticality applications [IBM21]. The act of dividing the host machine’s resources is called *partitioning* and each resource segment is called a *partition* [Rus00].

Popek et al. classify hypervisors into two types: (1) native or bare-metal hypervisors and (2) hosted hypervisors [PG74]. The former run directly on the host machine’s hardware and the later run on a conventional operating system. Hypervisors enforce *spatial isolation* in partitioning, which protects the code or private data of one partition not to be altered by any other partitions. Spatial isolation is achieved via address translation and assignment of a different memory location to each partition [Rus00]. Since a partition may run applications that require timeliness, the hypervisor may enforce *temporal isolation* which protects the timeliness of applications in one partition from being affected by the running of the other partitions [Rus00]. Hypervisors enforce the spatial and temporal isolation by utilizing hardware-supported virtualization extensions of modern processors that allow such mechanisms [RS19, SVLN13].

Throughout this thesis, we assume FNs of the FCP use hypervisors such as ACRN [ACR20] and PikeOS [KW07] to virtualize control applications and separate applications in static partitions using spatial and temporal isolation. Each static partition consists of several partition slices that are time slots to which the resources are assigned to the partition and is scheduled via static partition tables that capture the start time and finishing time of each partition slice.

1.2.1.3 Scheduling Policies

Scheduling is a method for assigning resources for completing activities (e.g., tasks or messages) [Wik21a]. The activity can be computation elements or data flows. The scheduling is carried out by a *scheduler* that implements a policy to achieve goals such as minimizing latency, minimizing wait-time, and maximizing throughput. Most schedulers use policies that are not real-time; thus, they are not suitable for real-time applications.

There are two approaches in the literature for scheduling: (i) time-triggered, such as non-preemptive static cyclic scheduling, also called timeline scheduling, and (ii) event triggered, such as preemptive periodic scheduling, e.g., using fixed, Rate Monotonic (RM) or dynamic, Earliest Deadline First (EDF) priorities [But11]. Static cyclic non-preemptive scheduling has been recommended for safety-critical applications [Kop11, SL03]. In this method, the scheduler repeats a static schedule determined at design-time with a given cycle, i.e., a hyperperiod (also called major cycle), see Sect. 1.3.2. The scheduler divides the time interval into time slots of equal lengths which can be allocated to perform activities by synchronizing their activation at the beginning of each time slot [But11].

The preemptive periodic scheduling approaches generate the schedule at runtime based on arrival times of periodic activities and their priorities [But11]. Researchers have shown how periodic servers can be used to handle aperiodic activities [But11], which are typical for non-critical Fog applications. Thus, the server acts similarly to any periodic activity and is scheduled by the same scheduling policy as the other periodic activities. The server prioritizes the completion of an activity on its arrival based on a given priority metric [BLAC05]. Once an aperiodic activity arrives, the server executes it within the limit of its capacity. The activity may be preempted several times by the termination of the server's time slot or higher-priority activities until its execution is completed.

Mixed-criticality applications require different scheduling policies depending on their time-criticality. Similar to related work, we use static cyclic scheduling (timeline scheduling) for critical hard real-time tasks since this is a scheduling policy suited for hard real-time applications in safety-critical areas. Fog applications are scheduled using a deferrable server approach [BLAC05, SLS95, SSL89]. To put together several scheduling policies, we use the *hierarchical scheduling* model [WZS$^+$14].

A hierarchical scheduling framework can be generally represented as a tree, or a hierarchy, of nodes, where each node represents a scheduling approach and resources are allocated from a parent node to its children nodes [SL03]. Any scheduling algorithm can be employed as a scheduling approach if it is able to provide real-time guarantees. Although hierarchical scheduling has been typically used in conjunction with scheduling policies that handle periodic activities, aperiodic activities can also be handled [LB10]. As mentioned, aperiodic activities model the dynamic Fog applications that migrate across FNs and are best-effort. Such Fog applications should be isolated from critical applications, e.g., via partitions that protect higher-criticality real-time applications, and are handled using best effort policies, e.g., resource reservation techniques [BLAC05]. The performance and predictability of these mixed-criticality applications can be analyzed using the results of real-time computing theory [But11, BLAC05, BD13].

1.2.1.4 Time-Sensitive Networking

Real-time and safety-critical areas use several specialized protocols such as CAN [Spe91], FlexRay [Con05], SAFEbus [HD92] and AFDX [Pic06]. The market for fieldbuses in the industrial automation area is highly fragmented and uses proprietary protocols such as ProfiNet [Comb] and EtherCAT [Coma]. Due to increasing demands for more bandwidth, interoperability and standardization, there is currently a trend to extend the well-known Ethernet protocol [Gro21] for real-time and safety-critical applications.

Time-Sensitive Networking (TSN) is a set of standards developed by the IEEE Time-Sensitive Networking Task Group of the IEEE 802.1 Working Group [IEE21b]. TSN defines mechanisms for the time-sensitive and dependable transmission of data over switched Ethernet networks. Table 1.1 presents the current set of standards and amendments in TSN[1].

The deterministic behavior of TSN is realized via: (1) shapers and schedulers and (2) stream reservation protocols. The former has been addressed in IEEE 802.1Qav (Forwarding and Queuing Enhancements for Time-Sensitive Streams), IEEE 802.1Qbv (Enhancements for Scheduled Traffic), and IEEE 802.1Qcr (Asynchronous Traffic Shaping), see Table 1.1. The later has been introduced in IEEE 802.1Qat (Stream Reservation Protocol), IEEE 802.1Qcc (Stream Reservation Protocol Enhancements and Performance Improvements), and IEEE 802.1CB (Frame Replication and Elimination for Reliability) where reliability requirements are addressed.

[1] References to the standards can be found via IEEE Xplore using their names.

Table 1.1: Standards and amendments in TSN

Standard	Definition
802.1Q-2005	Virtual Bridged Local Area Networks
802.1Q-2011	Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks
802.1Q-2014	Bridges and Bridged Networks
802.1BA-2011	Audio Video Bridging (AVB) Systems
802.1BA-2011/Cor 1-2016	Audio Video Bridging (AVB) Systems– Corrigendum 1: Technical and Editorial Corrections
802.1CB-2017	Frame Replication and Elimination for Reliability
p802.1Q-2018	Bridges and Bridged Networks (Project Authorization Request)

Amendment	Definition
802.1AS-2011	Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks
802.1Qav-2009	Forwarding and Queuing Enhancements for Time-Sensitive Streams
802.1Qat	Stream Reservation Protocol (SRP)
802.1Qca-2015	Path Control and Reservation
802.1Qbv-2015	Enhancements for Scheduled Traffic
p802.1AS-Rev	Timing and Synchronization for Time-Sensitive Applications (Project Authorization Request)
p802.1Qcc	Stream Reservation Protocol (SRP) Enhancements and Performance Improvements (Project Authorization Request)
p802.1Qcr	Asynchronous Traffic Shaping (Project Authorization Request)

Different traffic types have been introduced in TSN, such as Scheduled Traffic (ST) also called Time-Triggered (TT), Audio-Video Bridging (AVB), and Best Effort (BE), to support different timeliness requirements. ST is being sent based on schedules stored in the network switches using the IEEE 802.1Qbv (Enhancements for Scheduled Traffic) amendment. With the ST, TSN provides guarantees for traffic timeliness. In this thesis, we focus on the ST type which is the most suitable for real-time applications where stringent timing is required. However, the methods in this thesis can be used in conjunction with any traffic type where the latency can be bounded, that is, can handle real-time transmissions.

A TSN switch has a number of ports, a switching fabric, priority queues, gates, and a Gate Control List (GCL) (see Fig. 1.3 for more details of TSN switch internals).

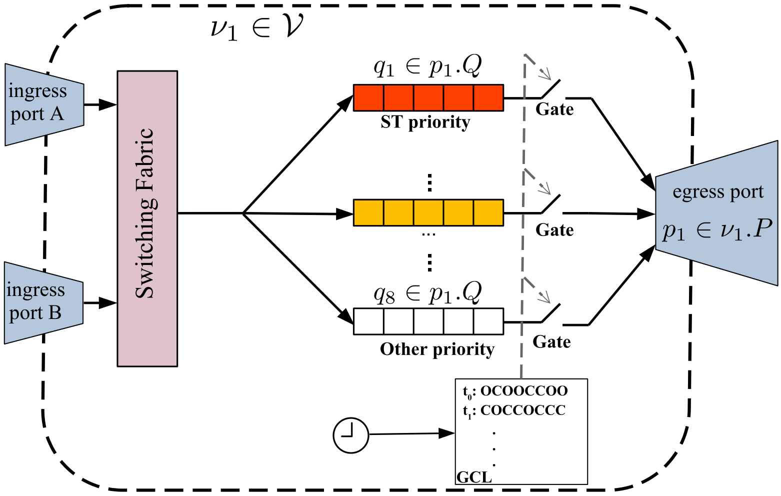


Figure 1.3: Schematics of TSN Switch internals [BP21a].

Each port composes an *ingress port* and an *egress port* which has a set of eight priority queues (according to the IEEE 802.1Q standard [IEE14]). The switching fabric receives traffic in the form of flows from the ingress ports and forwards each flow to the egress port which is determined by the predefined internal routing table. The egress port stores the flow that has a Priority Code Point (PCP) field in the frame header that specifies the priority in a relevant priority queue in First-In-First-Out (FIFO) order. According to the 802.1Qbv standard, the transmission of traffic from each queue is regulated by an associated gate that opens and closes based on a predefined GCL that 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 the gate is closed.

1.3 System and Application Models

1.3.1 Architecture Model and Example

An FCP consists of FNs, sensors (e.g., cameras, radars, etc.), and actuators (e.g., electric drives, robots, etc.), which we assume that they are connoted to each other via TSN. The TSN network consists of End-Systems (ESs) (which are the FNs, sensors, and actuators in the FCP) and network Switches (SWs) interconnected via full-duplex bi-directional physical links. The SWs can be stand-alone (typical in the industrial area) or integrated into the ESs (typical in automotive), see Fig. 1.4. In this thesis, an

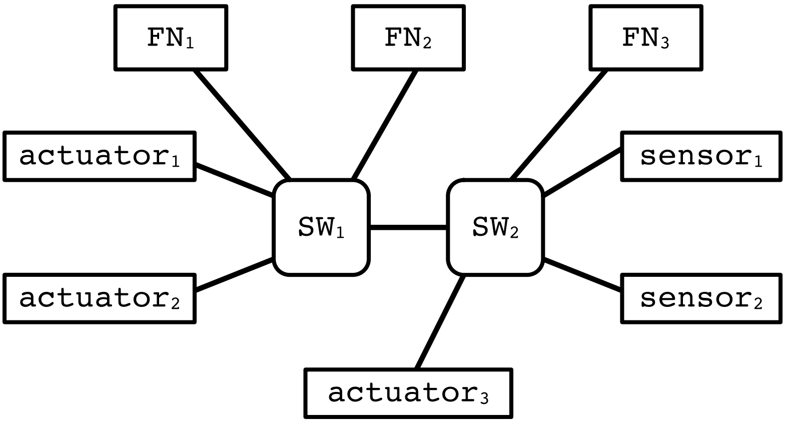


Figure 1.4: An example architecture model: Three Fog nodes are connected to five end-systems via two switches.

FCP is modeled as a directed graph for better understanding. The graph nodes are the ESs, and the graph vertices are the physical links.

A physical full-duplex bi-directional link is known with the nodes it connects (e.g., FN_1-ES_1), and has a specific speed. A sequence of such links is called a route which starts with a link originating from a talker ES and ending with a link to a listener ES. A message in TSN is forwarded through an associated route.

In the example given in Fig. 1.4, three FNs, two SWs, and five ESs (i.e., two sensors and three actuators) are connected to each other via physical links. In this example, the FN_1 has a dual-core CPU. Additionally, a possible route is $[FN_1\text{-}SW_1, SW_1\text{-}SW_2, SW_2\text{-}actuator_3]$.

1.3.2 Application Model and Example

The application areas addressed in the thesis, such as automated manufacturing, critical infrastructures, smart buildings, and smart cities, are typically safety-critical and real-time, requiring guaranteed non-functional properties, such as timeliness, reliability, availability, safety, and security.

Aligned to other research initiatives [PZB$^+$21], in this thesis, we envision that Fog Computing will be used as an architectural means of implementing such systems. In such a context, an FCP will host mixed-criticality applications, which may include: safety-critical control applications, real-time applications, and best-effort Fog applications.

In addition to control applications, an FCP will also host a variety of real-time applications, both hard and soft. The promise of using Fog Computing, which spans the continuum from things (machines) to Cloud, is that the platform will host a myriad of novel services that can unlock the new value, e.g., edge data analytics [PZB+21] that can be used to optimize the system-level operations. Such services are developed as best-effort applications which do not need strict guarantees but would benefit from optimizing quality-of-service metrics such as average response time. There have been several application models proposed in the literature, depending on the periodicity and time-criticality of the applications [But11, Kop11].

As mentioned in the introduction, the vision is that the industrial control which is currently implemented via dedicated hardware and control software, will be implemented on an FCP as control applications, i.e., it will be virtualized. With such an approach, the challenge—addressed by this thesis—is to achieve the same level of dependability for the safety-critical control applications as the one taken for granted in current dedicated implementations. Our approach is to rely on methods and tools that statically configure at design-time the resources required by these applications, such that, in conjunction with redundancy, we can guarantee their timeliness and dependability requirements.

There is a plethora of application models proposed both in the real-time computing [But11, BLAC05] and the Fog Computing [PML+19, BMNZ14] communities. The different types of applications, with different criticalities and timeliness requirements, have to be modeled using different approaches. These models have to be able to capture timing properties such as Worst-Case Execution Times (WCETs), periods and deadlines for periodic real-time tasks, Quality of Control (QoC) for control tasks, as well as multiple types of constraints, such as data dependencies between tasks that communicate via messages, and resource sharing constraints [But11].

In this thesis, we use the periodic task model from the real-time computing literature [But11], which targets hard real-time applications, to model both the critical control applications and the real-time applications mentioned earlier. In the following, we provide the basics of this model, which is further detailed in Sect. 2.2.2 (for tasks) and Sect. 3.2.3 (for messages). Sect. 1.3.3.1 presents the timeliness properties used for these applications. In addition, Sect. 1.3.3.2 introduces the QoC metrics we use for modeling the control applications. Finally, the dynamic Fog applications are modeled with the aperiodic task model from real-time computing [But11], and the focus is on minimizing their response times.

A hard real-time application is periodic and it is characterized by a deadline that must be met. The hard real-time applications 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. Applications are typically modeled as interacting tasks that exchange messages. In this thesis, applications are modeled using a Directed Acyclic Graph (DAG) [But11], where the vertices represent tasks and edges represent network messages exchanging between the tasks, capturing the data dependency between the tasks. A data-dependent task is ready when all of its inputs have arrived and produces its outputs when it is terminated.

A control application samples the output of the dynamical system, calculates the deviation from the desired output and drives the deviation to zero by applying an appropriate control signal. Thus, a control application can be modeled with a DAG consisting of three tasks exchanging messages, see $Application_1$ in Fig.1.5. The source task, let's say $Task_1$ samples the dynamical system by using sensors and may process the captured data. A typical process for the source task is analog to digital conversion. Once the data becomes ready, the tasks send it as a message, i.e., $Message_1$ to the control task that is $Task_2$.

$Task_2$ receives $Message_1$ and calculates the control signal. This task utilizes various methods for the calculation and may be engaged with time-consuming calculation [OY02]. The implemented method for calculating the control signal is called the control law. The control signal is send as $Message_2$ to the sink task that is $Task_3$. $Task_3$ may process the input data and exert the control signal to the dynamical system using the connection with the actuators. A typical process for $Task_3$ is digital to analog conversion.

A periodic hard real-time task is characterized by a WCET, which is a theoretical upper bound on the task's execution [EES$^+$03], a period and a hard deadline. We assume that we know the WCETs on the cores of FNs where the task is considered for mapping. A message is characterized by a period inherited from the sending task, and a size in bytes. The messages that are sent outside of an FN use TSN and thus are transmitted via flows, which encapsulate the payload into frames. The flows are transmitted using Ethernet frames. Hence the flow will be split into multiple frames when the size of the message exceeds the Maximum Transmission Unit (MTU) of 1,542 bytes [Gro21].

Fig. 1.5 shows an example application model with two real-time applications. The $Application_1$ represents a control application consisting of three tasks. The $Application_2$ has four hard real-time tasks exchanging messages. Each flow is characterized with a tuple showing flow's size and period $\langle Size, Period \rangle$. Similarly, each tasks is characterized with a tuple showing task's WCET and period $\langle WCET, Period \rangle$.

1.3.3 Objectives

Engineering Fog-based systems is challenging, as it has to meet multiple competing design objectives. Some of the design objectives that need to be considered are: SWaP—referring to size, weight and power consumption [Hea02, NSL17, Zur05], performance— referring to, e.g., latency and throughput [Hea02, NSL17, Zur05], pre-

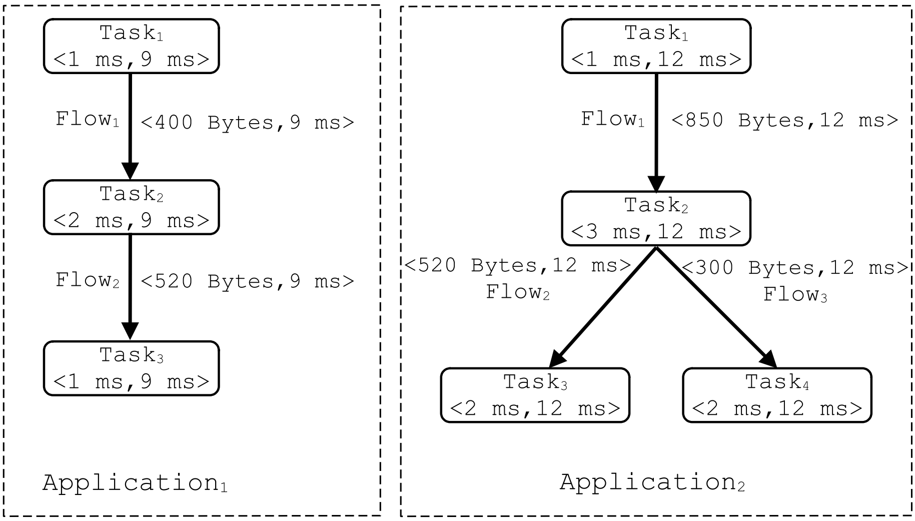


Figure 1.5: An example application model: $Application_1$ has three tasks exchanging flows. $Application_2$ has four tasks exchanging flows.

dictability—referring to the degree to which a correct prediction or forecast of a system's state can be made either qualitatively or quantitatively, which is important for guaranteeing timeliness properties [But11].

An important "umbrella" objective for safety-critical real-time systems is *dependability*, i.e., "the ability to deliver service that justifiably be trusted" [ALR$^+$01]. The main attributes of dependability are reliability, which represents the continuity of correct service, availability which represents the readiness for correct service, safety which represents the absence of catastrophic consequences on the users and the environment, integrity which represents the absence of improper system state alterations, confidentiality which represents the absence of unauthorized disclosure of information, and maintainability which represents the ability to undergo repairs and modifications [ALR$^+$01]. Most of these systems are typically developed as products on a competitive global market, where objectives related to as cost and time-to-market are paramount. Time-to-market refers to the necessary time to design and produce the system to the point it can be sold, and the cost objective captures, e.g., the unit cost, the cost to produce one copy of the product or the Non-Recurring Engineering (NRE) cost, the one-time engineering cost to design and develop the system [VG01].

This thesis focuses on the design metrics such as timeliness, QoC, and extensibility. The following sections describe in detail these design metrics.

1.3.3.1 Timeliness

Although real-time applications are characterized by a deadline which has to be met, the timeliness of a control system is also the result of metrics such as delay and jitter [Kop11, But11].

The delay, also referred to as response time, is the time elapsed between the ending and starting of an action [But11]. Thus, the delay of a task is the time elapsed between when it becomes ready for execution and when it is executed; and the worst-case response time of a task is defined as its maximum delay [But11]. Similarly, the delay of a flow is the time elapsed between when it started its transmission from the talker node, and when it is received in the listener node; and the maximum delay of a flow is referred to as the worst-case delay.

Jitter is the deviation from the true periodic timing of an event, e.g., a task execution and a flow transmission [But11]. It can be associated with the start time, the end time, and the duration of an event. It is also associated either among all instances or two consecutive instances. The predictability of a system is affected by the number of factors that impact the delay variations is minimized, i.e., when the jitter is low [But11].

To this end, an FCP having a real-time behavior must be configured such that all the actions of the system (e.g., task execution and flow transmission) are highly predictable [Kop11], including the delays that affect the actions meeting their deadlines and the jitter that affects the delay variation.

1.3.3.2 Quality-of-Control

A control application deals with the control of a dynamical system by driving it to the desired state. The control application samples the state of the dynamical system (using sensors), calculates the deviation from the desired state, calculates an appropriate control signal for driving the deviation to zero (using the control process), and applies the control signal (using actuators) [OY02]. Fig. 1.6 shows a simple control application, where Y is the current state of the system, R is the desired state, E is the deviation from the desired state, K is the control process, U is the control signal, W is disturbances, and X is the input to the dynamical system.

While designing the control process, for finding the suitable control law and tuning it, several parameters such as the damping ratio, the phase margin, and the gain margin (see [OY02] for more details) have to be determined. These parameters affect the accuracy and rapidity of the control application, in opposite directions. The trade-off between accuracy and rapidity of the controller, is called the Quality of Control (QoC)

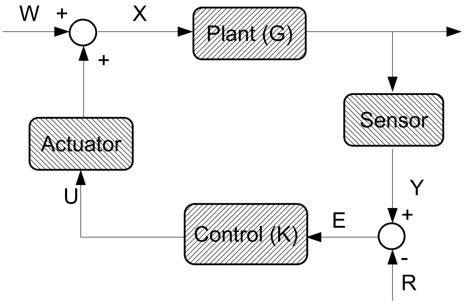


Figure 1.6: Schematics of a control application [BP21a]

(which is used interchangeably to mean “control performance”), see [Cer03] for more details.

The QoC of a control application 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 the 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. 1.7 shows the step-response of a sample control loop where these associated parameters are depicted.

Nevertheless, the QoC can also be impacted by the implementation of the control application and the changes in runtime due to the discrete time nature of the real-time systems.

A control application is typically implemented as periodic real-time tasks doing computation and exchanging messages whose periods depend on the dynamical system. Analyzing the bandwidth of the closed-loop dynamical system returns an interval from which the task periods can be chosen [AW97], see Sect. 2 for more details. The shorter the period, the faster the controller is able to respond to the disturbances, the larger the phase margin (resulting in a better QoC), and the more computational power is required (which is a bottleneck on real-times systems where the resources are constrained).

Additionally, the task timing is another source that impacts the QoC. The task timing refers to (1) the delay between the sampling task and the actuation task, and (2) jitter in the task execution and the message exchange. The former directly decreases the phase

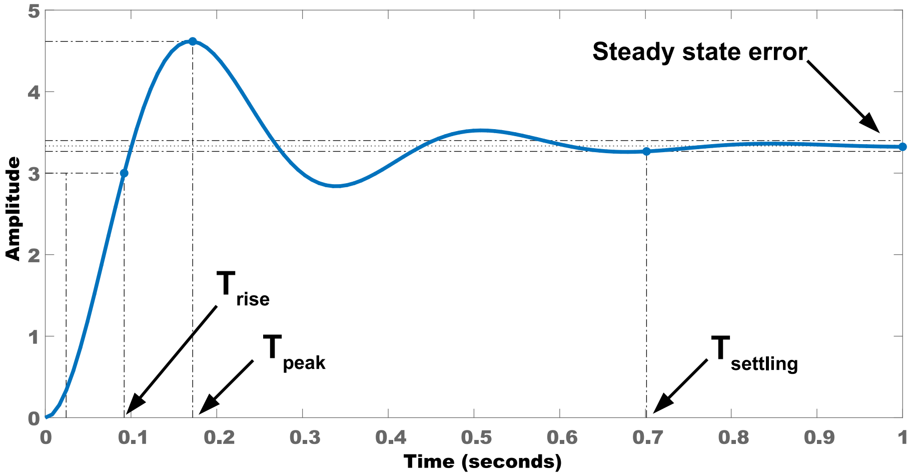


Figure 1.7: Step response of a sample control loop [BCP20]

margin of the controller [OY02], leading to worse QoC, whereas the later's effects on the controller are less obvious to analyze.

The various aspects of the QoC can be captured in a cost function. A common choice [SLSS96] is to use a quadratic cost function of the form

$$J = \int_0^\infty \left(x^T(t) Q_1 x(t) + u^T(t) Q_2 u(t) \right) dt, \tag{1.1}$$

where the weighting matrices Q_1 and Q_2 tell how much deviations in the different states $x(t)$ and the control output $u(t)$ should be penalized. By proper tuning of the cost function, the desired transient and steady-state behavior can be achieved in the control design. The same cost function can also be used to evaluate the performance of the controller under non-ideal circumstances. A larger value of the cost J then means that the response is more sluggish or more oscillatory, typically increasing the settling time of the system.

However, the calculation of the QoC is possible via a simulation of the control behavior. Thus, several tools such as Jitterbug [LC02], JitterTime [CPBM19], and TrueTime [HCÅ02] are proposed to simulate the control behavior.

Jitterbug calculates the QoC assuming a fixed or a random jitter applied to inputs and outputs of the control function. Additionally, it can be used to design controllers concerning the stability margin. JitterTime calculates the QoC assuming a given schedules that capture the timing of control flow transmission and control task execution. Also, it can be employed to analyze the sensitivity of a control function to delay and jitter.

TrueTime simulates the execution of a control application based on a given schedule tables, and analyzes the control output.

1.3.3.3 Extensibility

Applications often need updates over the time due to the changes in their environment, e.g., design goals, machines they control, and the infrastructure [RLL94], the need to update security polices, and the need for improved functionalities. In safety-critical areas, it is costly to re-design the system architecture and/or change the configuration, because this often requires costly verification, validation and certification. The solution is to design an architecture and/or configuration for evolvability [RLL94], i.e., meeting the current applications' requirements and accommodating future changes to the applications. However, evolvability also comes with costs. The effective evolvable architecture is the one that adapts to the changes with a reasonable cost [IM94].

In this thesis, we aim for a specific type of evolvability, i.e., *extensibility* which is the capability of extending a solution for future upgrades [BFG+95]. The FCP, in this case, is designed for hosting hard real-time applications and envisioned to enable upgrades. The upgrades are assumed to accommodate dynamic Fog applications and add future critical control applications.

To this end, extensibility is formulated as the maximum action a schedule (that captures the timing of actions, such as tasks or messages) can accommodate for a new action within a certain period [WCP+05, ZYS+09]. Thus, the extensibility for tasks aims at maximizing the serviceable execution time for new independent tasks, and the extensibility for flows aims maximizing the serviceable transmission time for new independent flows. Having a prior knowledge on the specification of the new tasks and flows, the schedules can be optimized for extensibility [WCP+05, ZYS+09, PEPP04].

However, in this thesis we assume that there is no prior knowledge on the specifications of the dynamic Fog applications, and the extensibility is defined in a way that provides general solutions independent of the new applications' specifications.

1.4 Systems Engineering Decision Tasks and Related Work

The goal when engineering a system is to deliver an implementation that accomplishes the objectives set by the stakeholders, including those objectives associated with the creation, production, and disposal of the system [BM16]. Thus, identifying the stakeholders and their objectives is the first phase in systems engineering [BM16]. These objectives address the full life cycle of the system, including the design, integration, and management of the system, resulting in a set of requirements for decision making.

The main systems engineering phases, considering the decision-making tasks, are as follows [BM16]:

- *Conceptual Design* covers defining the base subsystems, identifying a suitable technology for a subsystem, and reviewing the existing hardware and software for the chosen technology.
- *Preliminary Design* deals with specifying the physical architecture, defining the required functions, allocating functions to resources, and identifying the validation and test structure.
- *Full-Scale Design* means specifying the details of each component, defining the configuration of each component, and identifying the items that need to be manufactured.
- *Integration and Qualification* covers planning tests, preparing equipment and facilities for testing, implementing the activities.
- *Product Refinement* is related to identifying the components that need to be improved, specifying the technology needed for improving a component, deciding on the refinement of a component.

We can distinguish between design-time and runtime decision-making. Most of the decisions are made at design-time, interchangeably called “offline decision making”, i.e., before the release of the system to the market [MSMB11]. However, many application areas require the flexibility to adapt to changing situations [McC96], thus making a decision at runtime is a necessity in these areas. In addition, recovery from failures may involve runtime decision-making on how to mitigate them.

Traditional control systems are engineered using design-time decision-making due to safety assurance considerations [GK19]. Certification of safety-related systems requires pre-release evidence that the safety risks have been identified and the appropriate mitigation measures have been implemented [GK19]. However, novel Industry 4.0 applications need adaptable and flexible implementations to meet dynamic demands. Therefore, the Fog-based implementations should be evolvable, allowing both computation and communication to be reconfigured to address changing needs and to decrease costs and resource usage [PRGS18].

Hence, in the context of the Fog-based systems envisioned in this thesis, we consider that the decisions related to the configuration of critical control applications (that are statically allocated to FNs) are performed at design-time, aiming to guarantee their timeliness and dependability requirements. The design-time decision tasks investigated by this thesis are: (1) determining the partitions for isolating the mixed-criticality tasks, (2) mapping the tasks to FNs and to the cores of the FN, (3) scheduling tasks of applications on cores of the platform, and (4) synthesizing the GCLs for message in TSN. These decisions optimize several metrics such as schedulability, QoC, and extensibility.

The configuration decisions related to the dynamic Fog applications (that migrate across FNs of the FCP) are performed at runtime. Examples of these runtime decisions, are: where to deploy these applications (on which FN), on which core of the FN to execute these applications (implemented as tasks), how to schedule tasks of these applications to execute, which route their messages to take, and how to schedule flows of these applications for transmission. In this thesis, we consider the following runtime decision tasks: (1) placing the tasks to FNs and to the cores of the FN, (2) determining slacks for scheduling tasks to run, and (3) determining slacks for scheduling flows to transmit.

In the next sections, the related work to these design tasks is presented.

1.4.1 Related Work

The related work is split into four subsections, addressing the main decision tasks considered.

1.4.1.1 Task Scheduling and Mapping, and Partitioning

The problem of task scheduling and mapping has been a well-studied topic within the real-time community, and different scheduling algorithms have been proposed [But11, BLAC05, Kop11, Mal09, SSL89]. For example, in [LT01], a fuzzy model is proposed for scheduling real-time tasks with imprecise deadlines and processing times. The model focuses on the optimal priority assignment to tasks which are then scheduled using EDF algorithm [But11].

Researchers have also proposed task scheduling algorithms for distributed systems hosting critical applications [CSB90, AS99, CSE14, SWP90]. In [THM02], the authors have proposed a heuristic approach for achieving the minimum execution time for critical tasks. In this approach, tasks are prioritized based on the upward rank value to achieve the earliest finish time. The work in [KJ09] proposes using Simulated

Annealing to maximize the throughput via minimizing makespan and communication cost, maximizing CPU utilization, and reducing the runtime.

There is also much work on optimizing task and mapping schedules. For example, in [OA09], the authors propose a genetic algorithm for optimizing task and mapping schedules for the shortest scheduling length and balanced workload of cores. The proposed algorithm employs the task duplication technique to overcome the overhead for communication between the tasks. The work in [THM02] proposes a heuristic approach for task mapping and scheduling, aiming to achieve the minimum total execution time for tasks. In this approach the summation of upward and downward rank values is used to prioritize tasks for scheduling. Energy optimization of task schedules and mapping has also been addressed by researchers such as [YXC02]. In this work, an Integer-Linear Programming (ILP)-based solution is proposed which first, maps the tasks to the processors such that the opportunities for lowering the voltage for processors increases, and second, schedules the tasks on the mapped processors to achieve the minimum voltage on processors.

The problem of scheduling mixed-criticality tasks separated with spatial and temporal isolation has been addressed by several researchers [WS12, MH18, JX07, BD13]. In this approach, partitions with different criticality levels separate the applications. Resources are allocated to the partitions based on the criticality levels, which assures resource availability and accessibility for critical applications with high priority. These applications would have guaranteed dependability by promising separation. A presentation of scheduling in mixed-criticality systems, which also considers partitioning, is presented in [BD13]. For example, Tamas-Selicean et al. [TSP15a] propose a method in which different Safety Integrity Levels (SILs) are assigned to the applications. In this method, applications with the same SIL are mapped to a single partition. Each partition is allocated several time slots on a processor to execute respective tasks. Concerning this method, the approach can provide a partition for each control application.

The authors in [MEA$^+$10] proposes an EDF-based algorithm that provides temporal isolation among mixed-criticality tasks. The algorithm prioritizes the scheduling of tasks based on their criticality level and uses the slacks that are redistributed in each iteration for less critical tasks. The work in [LCSW14] proposes an algorithm based on EDF for scheduling mixed-criticality tasks. The algorithm decides the scheduling at design-time assuming the worst-case execution time of high criticality tasks. It also reclaims the slacks caused by earlier termination of tasks, and schedules the less-critical tasks at runtime. [SZ13] proposes an Early-Release EDF algorithm for scheduling mixed-criticality tasks, which implements a slack reclaiming approach in scheduling less-critical tasks. In the work, less-critical tasks are assumed to have multiple periods with a minimum service requirement that is ensured by the largest period.

1.4.1.2 TSN Scheduling and Routing

There is a lot of work on routing and scheduling for TSN. Researchers have addressed the routing and scheduling problem 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 [DN16], 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. Several scheduling and routing solutions employing greedy-based heuristic approaches have also been proposed [WSJD15, AHM18]. Wisniewski et al. in [WSJD15] propose a less resource demanding approach that is possible to be implemented on industrial equipment on the field floor. This method focuses on increasing the flexibility of static schedules. The method can be integrated with discovery and auto configuration mechanisms allowing the integration to the field level. The authors in [AHM18] aim to generate joint network routing and communication scheduling that are fault-tolerant, within a reasonably short time. The routing problem is implemented as an iterative path selection problem starting from a fully connected network to the solution where costly links are removed.

Other heuristic solutions are also employed to solve such a scheduling and routing problem. For example, the work in [AHG20] proposes a hybrid genetic algorithm for the communication scheduling and network routing to find a near-optimal solution in a reasonable time, and also optimize the bandwidth to let more less-critical traffic transmitted. The work in [PTO19] proposes a heuristic list scheduler, where multi-cast traffic and application distribution are allowed, and bandwidth is optimized. The same problem is addressed in [PO18] where a genetic algorithm is employed, and in [GZRP18], where multiple traffic types are considered.

The use of SMT solvers for the communication scheduling is first proposed in [Ste10]. 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. Researchers have also proposed using mathematical programming for TSN scheduling and routing [COCS16, COSS17, SCS18a, SCS18b]. Craciunas et al. in [COCS16] 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 [COSS17], which introduces windows in GCLs and maps the frames to them. Another SMT model based on “array theory encoding” is proposed in [SCS18a], 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-off between the GCL length and run-time is well studied in [SCS18b] where the authors review the TSN standards and present a reference model for traffic scheduling in TSN.

The problem of scheduling and routing in TSN is extended for the benefit of other applications. For example, in [PRCS16], the authors combine traffic scheduling and network routing problems to achieve the minimum delay for AVB traffic. [MAS$^+$18] focuses on the message routing and scheduling of control applications to protect them from instability. It proposes a method for routing and synthesizing GCLs with careful consideration of the non-determinism of message, which guarantees worst-case message delay and jitter that control applications can tolerate. The traffic scheduling combined with task scheduling is studied in [CO16], where an SMT solver is employed to schedule network messages and tasks on a networked computation platform which is equipped with time-triggered network. The problem of increasing the resilience of the control applications to malicious interference is addressed in [MAS$^+$19], where the authors aim to increase the resilience of the control applications to malicious interference. Park et al. in [PSS19] propose a generic algorithm approach to schedule the communication in TSN where preemption is allowed. The proposed algorithm shows increased reliability in the generated solutions.

1.4.1.3 Configuration optimization for QoC

There is already much work on various topics related to Fog Computing, see surveys such as [YLH$^+$18, MKB18, MNY$^+$18]. However, the work so far addresses quality-of-service for applications that are not safety-critical and real-time. None of the approaches for resource management [AYY15] (that configure the FCP resources) are applicable to configuration for QoC. Thus, the QoC for control applications in the Fog is still an open issue. Nevertheless, there is much useful work in the literature that tackles the problem of degradation of control applications [ZHY16, ZSWY17, JX07].

Researchers propose several approaches (such as temporal and spatial separation of control tasks, virtualization of PLCs, scheduling of control tasks, co-design of control applications) that guarantee non-functional properties of control applications [SSS17, ZBP01, WS12, MH18]. The presented approaches are well studied and categorized into a category for platform configuration and a category for the integration of applications. Good performance for control applications will be ensured if both the applications and the platforms are configured.

The platform configuration aims to ensure the resource allocation to control applications and protecting their execution from interrupting by less-critical applications. This is achieved by separation and isolation of applications regarding their criticality levels, see Sect. 1.4.1.1. On the other hand, the integration of the applications in the platform affects their functional and non-functional properties. The co-design of control applications configure them at the integration level to achieve the highest performance. The co-design approach takes the platform characteristics into account while designing the application to have good integration with the platform.

There has been several much work in the area of co-design of control and real-time [BI07, XABC17, MYV$^+$04, PYKL11, BPZ02, SSS17, ZDN17] which has tackled the design of controllers and scheduling of the control tasks with respect to the control performance. The co-design procedure involves designing control applications such that the controller is robust against degradation due to the scheduling of the tasks. Co-design and scheduling concerning QoC for control applications are proposed in the seminal work of Seto et al. [SLSS96]. The authors optimize the period of control applications concerning the QoC and schedulability of the tasks.

Chwa et al. [CSL18] propose a co-design and scheduling method to maximize QoC for control applications. The authors assign a sampling period, and a maximum number of consecutive deadline misses as parameters for each task concerning system stability. Then, the tasks are scheduled concerning the parameters without compromising system stability and also with efficient use of resources. Samii et al. [SCEP09] present an approach in which a controller is synthesized for each plant, and the control tasks are scheduled concerning the priority of the tasks. In this work, the scheduling is based on the cost of control function, which aims to consider the maximum QoC for all the control applications. The same approach concerning co-design and scheduling of control applications is used in [MSZ11]. A similar co-design approach is presented by Cervin et al. [CEBÅ02]. In this work, the scheduler uses feedback from execution time and also feed-forwards the workload along with the cost of control to achieve the best QoC. Besides, control task parameters such as periods are changed with the feedback from execution time. The approach can compensate the impact of jitter on the QoC. In the work [RHS97], a heuristic algorithm is used to derive the periods and deadlines of the tasks, and end-to-end response of the control loop. The assigned parameters are assessed in a simulation that schedules and executes the tasks.

Another co-design approach is considered in [SGM$^+$16]. The proposed method gets feedback of delay and jitter in the execution of tasks from the scheduler and feeds it to the control applications. The controllers take the feedback and adjust the control outputs to compensate for the delay and jitter impact and to maximize the QoC. The feedback from the task scheduler is also used to predict the jitter and delay.

The performance of a control application is evaluated by measuring various parameters such as settling time, rise time, overshoot, offset error, etc. Various computational methods have been introduced to evaluate the performance of a control application [SR61, Ell66, BBC17]. Furthermore, related work has also investigated the impact of the virtualization of control applications (also in Cloud Computing) [GITJ14, CSM16, GMS^{+}15].

The problem of QoC analysis and schedulability has also been addressed in [ÅCES00, XCÅ16, CEBÅ02]. In the proposed approaches, the task scheduler schedules the tasks concerning the QoC of control applications. In the work by Schneider et al. [SGMC12], the QoC measurement is embedded in the task scheduler with allowed preemption. The scheduler is capable of handling mixed-criticality applications as well. Mahmoud et al. [MH18] use an optimization algorithm to derive timing constraints of control tasks, such as the task periods, to achieve maximum QoC for the control applications concerning the schedulability of the control tasks.

In [TG11], a feedback scheduling framework is developed to schedule control tasks such that the QoC is maximized for control applications and to adjust workload constraints. The QoC measurement is embedded in the task scheduler. The scheduler gets delay and jitter feedback to change the period of the tasks concerning the QoC and workload management. The same approach is used in [SGAN^{+}16] to schedule tasks concerning the QoC with feedback from scheduling. In this work, the period of tasks is changed regarding the feedback. Eker et al. [EHÅ00], propose a similar method. The method uses feedback from the scheduler to assign the period of control tasks. The period assignment provides good control performance along with optimizing the resource allocation out the tasks. Cha et al. [CJK16] propose a method for scheduling of control tasks which determines the deadline and period of the tasks for achieving maximum QoC. The method optimizes the QoC of the control applications and resource utilization. In the approach presented in [XCÅ18], the task scheduler guarantees bounded delay and jitter in the execution of the control application while the approach guarantees that the control application is still stable in the presence of bounded delay and jitter.

The control performance is not only affected by the scheduling of tasks but also affected by the scheduling of messages in the network. On one hand, researchers have addressed the configuration of communication aiming at increased control performance [WS12, HZ19, Son09], but very few work addresses TSN. On the other hand, there is much work on routing and scheduling for TSN, but none considers the QoC.

Only a few of the works concerning the scheduling of control tasks consider Deterministic Ethernet, such as TSN [MAS^{+}18]. For example, [MAS^{+}18] focuses on the routing and scheduling of messages of control applications in Deterministic Ethernet 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. The proposed method 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. Other initial investigations in [MAS^{+}18, BZP20] address the QoC and consider the particularities of TSN but use a simplified model for control applications.

1.4.1.4 Configuration Optimization for Extensibility

There has already been much work on designing for evolvability and extensibility in computing systems such as [GRS96, RLL94, RRW+03] aiming to enable future upgrades and changes for the real-time systems. However, extensibility for mixed-criticality systems is still an open problem. Since these systems host safety-critical applications, any changes in the system configuration requires re-certification [KZ09]. Nevertheless, there is much work in the literature that targets extensibility for mixed-criticality systems that separates applications with different criticality levels and provides static configuration for high-critical applications.

Pop et al. [PEPP04] propose an incremental scheduling algorithm for embedded systems which aims at facilitating applications with hard deadlines. This approach considers a system with tasks that have already started and generates extensible schedules for adding specific future tasks considering that the existing tasks should be disturbed as little as possible. In this work, the authors use the idle time slots of the schedules in order to provide extensibility. For the evaluation, an extensibility metric is defined as the distribution of the idle time slot profiles in the schedules concerning the future task sets.

Optimizing schedules for extensibility has been addressed in [ZYS+09, WCP+05] with different benefits. The approach presented in [ZYS+09] targets robust task scheduling in distributed systems concerning the changes in task requirements. In this work, the notion of extensibility is used for robustness. The extensibility metric is defined as the weighted sum of each task's execution idle time over its period. Zheng et al. [WCP+05] propose a mathematical modeling approach for extensible scheduling to accommodate additional tasks. In this work, the extensibility metric is defined as the maximum execution time a schedule can accommodate for a new independent task with a certain period. This definition distributes the idle time among all tasks and targets all variations of future task sets, i.e., no prior specification of future tasks is required.

Guo et al. [GGZ+12] propose a method based on ILP that decides the mapping of mixed-criticality applications to the cores of a computing platform used in automotive. The method is able to calculate and optimize solutions for extensibility. The authors also propose a formal and quantitative definition of extensibility that they call "flexibility".

The authors in [BKA+20] propose an extensible scheduling algorithm for critical applications in an FCP. The proposed algorithm employs a heuristic approach that provides well-distributed slacks in the schedules of high-critical applications, which can be used for scheduling future critical applications. The work in [WHL$^+$19] focuses on scheduling messages in TSN networks in automotive where dynamic messages with less-criticality are needed to be scheduled with ones of high criticality (that e.g., control engine); and optimizing the schedules to host more dynamic messages.

[MWTP$^+$13] proposes an approach based on Mixed Integer-Linear Programming (MILP) and Genetic Algorithms that aims to map the applications to the processing elements, separate the mixed-criticality applications using partitioning, and schedule tasks and messages of the applications. The approach considers several objectives for optimizing the solution, including extensibility. To this end, the extensibility is formulated as the distribution of slacks in the schedules. Similarly, Zho et al. [ZZZ$^+$13] propose a MILP-based approach that uses the worst-case response time analysis to decide on the task allocation, the signal to message mapping, and the assignment of priorities to tasks and messages. Although the approach focuses on minimizing latency, it can optimize solutions for maximizing slacks that can be used for accommodating future applications.

1.5 Thesis Overview and Contributions

During my Ph.D. studies I have published and submitted the following articles:

- M. Barzegaran, A. Cervin, and P. Pop, "**Towards quality-of-control-aware scheduling of industrial applications on fog computing platforms**," In Proceeding of the Workshop on Fog Computing and the IoT. ACM, pp. 1–5, 2019. [BCP19]

- A. Cervin, P. Pazzaglia, M. Barzegaran, and R. Mahfouzi, "**Using JitterTime to analyze transient performance in adaptive and reconfigurable control systems**," In Proceeding of IEEE International Conference on Emerging Technologies and Factory Automation. pp. 1025-1032, 2019. [CPBM19]

- 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. [BCP20]

- M. Barzegaran, N. Desai, J. Qian, K. Tange, B. Zarrin, P. Pop and J. Kuusela. "**Fogification of electric drives: An industrial use case**," In Proceeding of IEEE International Conference on Emerging Technologies and Factory Automation. Vol. 1, pp. 77-84, 2020. [BDQ$^+$20]

- M. Barzegaran, B. Zarrin, and P. Pop. “**Quality-of-control-aware scheduling of communication in TSN-based fog computing platforms using constraint programming**,” In Workshop on Fog Computing and the IoT, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, pp. 1-4, 2020. [BZP20]

- P. Paul, B. Zarrin, M. Barzegaran, S. Schulte, S. Punnekkat, J. Ruh, and W. Steiner, “**The FORA Fog Computing Platform for Industrial IoT**,” In Information Systems, Elsevier, vol. 98, pp. 101727, 2021. [PZB$^+$21]

- M. Barzegaran, V. Karagiannis, C. Avasalcai, P. Pop, S. Schulte and S. Dustdar, “**Towards Extensibility-Aware Scheduling of Industrial Applications on Fog Nodes**,” In Proceeding of IEEE International Conference on Emerging Technologies and Factory Automation, pp. 67-75, 2020. [BKA$^+$20]

- I. Murturi, M. Barzegaran and S. Dustdar, “**A Decentralized Approach for Determining Configurator Placement in Dynamic Edge Networks**,” In Proceeding of IEEE International Conference on Cognitive Machine Intelligence, pp. 147-156, 2020. [MBD20]

- M. Barzegaran, N. Reusch, L. Zhao, S. Craciunas, and P. Pop. ”**Real-Time Guarantees for Critical Traffic in IEEE 802.1Qbv TSN Networks with Unscheduled End-Systems**,” In arXiv, 2021. [BRZ$^+$21]

- M. Barzegaran and P. Pop, “**Communication Scheduling for Control Performance in TSN-Based Fog Computing Platforms**,” In IEEE Access, vol. 9, pp. 50782-50797, 2021. [BP21a]

- M. Barzegaran and P. Pop. “**Extensibility-Aware Fog Computing Platform Configuration for Mixed-Criticality Applications**”, Submitted to IEEE Transactions on Services Computing, 2021. [BP21b]

- M. Barzegaran, N. Desai, J. Qian and P. Pop, “**Electric Drives as Fog Nodes in a Fog Computing-based Industrial Use Case**,” Submitted to IET Journal of Engineering, 2021. [BDQP21]

- J. Qian, M. Barzegaran, and P. Pop “**Decomposing Deep Training Solutions on Fog Computing Platforms**,” To be submitted to ACM/IEEE Symposium on Edge Computing, 2021. [QBP21]

This thesis consists four of the listed articles as: [BCP20] in Chapter 2 (Paper A), [BP21a] in Chapter 3 (Paper B), [BDQP21] in Chapter 4 (Paper C), and [BP21b] in Chapter 5 (Paper D). The contributions of the selected papers are described in the following subsections.

1.5.1 Paper A: Performance Optimization of Control Applications on Fog Computing Platforms Using Scheduling and Isolation

This paper addresses mixed-criticality applications characterized by their safety criticality and time-dependent performance, which are virtualized on an FCP. We use partitioning and static-cyclic scheduling to provide isolation among mixed-criticality tasks and to guarantee their timing requirements. The temporal and spatial isolation is enforced via partitions, which execute tasks with the same criticality level. We consider that the tasks are scheduled using static cyclic scheduling.

We are interested in determining the mapping of tasks to the cores of the FNs, the assignment of tasks to the partitions, the partition schedule tables, and the tasks' schedule tables, such that the QoC for the control tasks is maximized and we meet the timing requirements for all tasks, including tasks with lower-criticality levels. We are also interested in determining the periods for control tasks to balance the schedulability and the control performance. We have proposed a Simulated Annealing metaheuristic, which relies on a heuristic algorithm for determining the schedules and partitions, to solve this optimization problem.

Compared to the related work, the main contributions of this paper are as follows. We have taken into account the virtualization typically used in FCPs, which results in the use of partitions to separate mixed-criticality functions. Our method considers the partitioning and we have proposed a heuristic to determine the assignment of tasks to partitions. The method captures a range of periods for control applications that can be used to implement their tasks, determines the periods of control tasks to trade-off QoC and schedulability, and considers the preemption of tasks to make static schedules more flexible. In addition, we also consider a realistic model of control applications and provide accurate measure of QoC which simulates the behavior of a control application, handling realistic scenarios.

1.5.2 Paper B: Communication Scheduling for Control Performance in TSN-based Fog Computing Platforms

This paper focuses on real-time control applications that are implemented using FCPs and modeled as a set of real-time flows. We are interested to synthesize the GCLs for messages such that the 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++.

Compared to the literature, 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.

1.5.3 Paper C: Extensibility-Aware Fog Computing Platform Configuration for Mixed-Criticality Applications

In this paper, we consider that critical control applications and Fog applications share an FCP. Critical control applications are implemented as periodic hard real-time tasks and messages and have stringent timing and safety requirements, and require safety certification. Fog applications are implemented as aperiodic tasks and messages and are not critical. We formulate an optimization problem for the joint configuration of critical control and Fog applications, such that (i) the deadlines and QoC of control applications are guaranteed at design-time, (ii) the configuration is extensible and supports the addition of additional future new control applications without requiring costly re-certification, and (iii) the design-time configuration together with the runtime Fog resource management mechanisms, can successfully accommodate multiple dynamic responsive Fog applications. We evaluate our approach on several test cases assuming scenarios for hosting both Fog applications and future critical control applications. The results show that our approach generates extensible schedules which enables FNs to handle Fog applications with a shorter response time and a larger number of future control applications.

The contributions of this paper are as follows. We motivate the need for a novel configuration optimization approach for mixed-criticality applications running on an FCP. We assume that the platform uses partitioning to enforce the spatial and temporal isolation between applications with different criticalities. We use a hierarchical scheduling model that can accommodate multiple scheduling policies, targeting the different time-criticality requirements of applications. The critical control applications are scheduled using static cyclic scheduling (i.e., they are time-triggered) and the resources of the Fog applications are allocated at runtime via fixed-priority servers that are dimensioned at design-time jointly with the critical application configurations. We consider that the critical control applications use ST for their flows, implemented via IEEE 802.1Qbv, which defines a Time-Aware Shaper (TAS) mechanism that enables the scheduling of flows based on a global schedule table. The flows of the Fog applications use Strict Priority (SP) flows that are sent with lower priority in the gaps of the ST traffic schedule tables.

We propose a CP-based optimization strategy to synthesize such optimized configurations. At design-time, the configurations consist of decisions on the mapping of critical control tasks to the cores of the FNs, the routing of critical control flows, the schedule tables for critical control tasks and flows, the slack in these schedule tables to increase their flexibility, and the period and budget of the fixed-priority servers that allocate resources at runtime to the Fog applications. At runtime, our approach handles the migration of Fog tasks to the FNs that have resources for their execution, the scheduling of Fog tasks on the servers and of flows on TSN.

1.5.4 Paper D: Electric Drives as Fog Nodes in a Fog Computing-based Industrial Use Case

As one of the main components in industrial applications, electric drives control electric motors and record vital information about the respective industrial processes. Developing electric drives as FNs within an FCP, brings new offerings such as programmability, analytics and connectivity increasing their added value. In this paper, we use the FORA Fog Computing Platform reference architecture to implement electric drives as FNs, which we call “fogification”. We have designed our fogified drive architecture and its components using AADL. The design process was driven by high-level requirements that we have elicited. We have used the fogified drive architecture to implement an industrial conveyor belt use case where electric drives are the key components and evaluated the architecture. We have evaluated the fog-based use case design on several Key Performance Indicators (KPIs). The KPIs were used for evaluating our proposed fog-based drive design, showing its advantages over the current drive architecture.

In this paper, our contributions are as follows. We propose a new design for electric drives as FNs using the FORA FCP reference architecture from [PZB^{+}21] and model the fogified drive architecture using AADL. Additionally, we implement a realistic industrial application that uses electric motors to drive conveyor belts using the proposed fogified drive architecture. We identify the Fog-based drive requirements to drive the design process. We use the Fog-based drives to develop a solution for our use case, and propose several KPIs, which are used to evaluate the Fog-based solution.

CHAPTER 2

Paper A: Performance Optimization of Control Applications on Fog Computing Platforms Using Scheduling and Isolation

In this paper, we address mixed-criticality applications characterized by their safety criticality and time-dependent performance, which are virtualized on a Fog Computing Platform (FCP). The FCP is implemented as a set of interconnected multicore computing nodes, and brings computation and communication closer to the edge of the network, where the machines are located in industrial applications. We use partitioning and static-cyclic scheduling to provide isolation among mixed-criticality tasks and to guarantee their timing requirements. The temporal and spatial isolation is enforced via partitions, which execute tasks with the same criticality level. We consider that the tasks are scheduled using static cyclic scheduling. We are interested in determining the mapping of tasks to the cores of the fog nodes, the assignment of tasks to the partitions, the partition schedule tables, and the tasks' schedule tables, such that the Quality-of-Control for the control tasks is maximized and we meet the timing requirements for all tasks, including tasks with lower-criticality levels. We are also interested in de-

termining the periods for control tasks to balance the schedulability and the control performance. We have proposed a Simulated Annealing metaheuristic, which relies on a heuristic algorithm for determining the schedules and partitions, to solve this optimization problem. Our optimization strategy has been evaluated on several test cases, showing the effectiveness of the proposed method.

2.1 Introduction

We are at the beginning of a new industrial revolution, i.e., Industry 4.0, which is underpinned by a digital transformation that will affect all industries. Industry 4.0 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 Technology (OT) and Information Technology (IT), which 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 extra-functional properties, such as, real-time behavior, reliability, availability, industry-specific safety standards, and security.

OT uses proprietary solutions imposing severe restrictions on the information flow. IT such as Cloud Computing cannot be used at the edge of the network, where industrial machines are located, and where very stringent extra-functional properties have to be guaranteed [GVCL14]. Instead, a new paradigm, called *Fog Computing*, is envisioned as an architectural means to realize the IT/OT convergence. Fog Computing is a “system-level architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Things” [Ope21a]. 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 (see Fig. 2.1).

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 [BMNZ14]. The vision is to virtualize the control (which is implemented as control tasks running on a Fog Computing Platform) and achieve the same level dependability as the one taken for granted in OT. Several initiatives are currently working towards realizing this vision [PML$^+$19, HDNQ17].

The integration of computational and storage resources into the communication devices is realized in the Fog Node (FN). In many applications, including industrial automa-

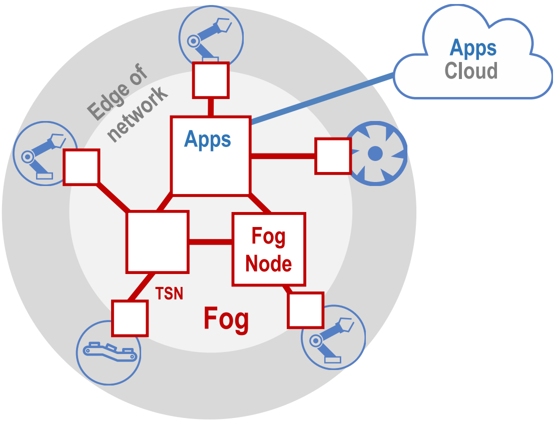


Figure 2.1: Fog Computing platform. Boxes represent fog nodes, connected with each other and to the Cloud; the thick lines are the network. Applications (Apps) run in the fog and Cloud.

tion 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. Researches have started to propose solutions for the implementation of FNs [BMNZ14, PML$^+$19] and fog node solutions have started to be developed by companies [PML$^+$19, HDNQ17, TTT21].

An FN is equipped with computational resources that allows the execution of applications and it is connected to a larger data processing facility like a Cloud environment. Regarding computation, we assume that the control tasks are running in an Real-Time Operating System using real-time scheduling policies (we consider static-cyclic scheduling in this paper), and the control applications are separated in different *partitions* enforced using hardware-supported virtualization, based on hypervisors, such as ACRN [ACR20] or PikeOS [KW07]. FNs could be connected to each others and to the machines through a deterministic communication solution, such as IEEE 802.1 Time-Sensitive Networking (TSN) [IEE21b], see Fig. 2.1. Such a Fog Computing Platform (FCP) allows to increase the spatial distance between the physical process and the FN that controls it, allowing the control functions can be executed remotely on the FN. However, the way the FCP is configured has an impact on the control performance of the control applications.

Given a set of mixed-criticality applications and an FCP, we are interested to determine an FCP configuration such that the Quality of Control (QoC) of control tasks is maximized and all the tasks meet their deadlines. Determining an FCP configuration means deciding on the partitions, the mapping of tasks to the FNs and partitions, the schedule table for tasks, the partition table for partitions, and the periods of control tasks. We do not address the scheduling of messages on the TSN network, which can be solved with approaches such as [CSOCS16] that achieve low latency and zero jitter.

2.1.1 Contributions

This paper shows that when control becomes virtualized, implemented as tasks on an FCP, the configuration of the FCP has a strong impact on the control performance. We formulate the FCP configuration as an optimization problem, and we have proposed a metaheuristic solution to solve it. Compared to the related work (see Sect. 2.7), which has addressed the scheduling of tasks to maximize QoC, we also optimize the partitioning, which is required in an FCP to provide isolation among mixed-criticality applications, decide on the mapping of tasks to partitions, consider the preemption of tasks to make static schedules more flexible, and determine the period of control tasks to trade-off QoC and schedulability of non-control tasks. In addition, we also consider a more realistic model of control applications and provide more accurate measure of QoC compared to previous work, see Sect. 2.4.

2.1.2 Outline of the Paper

In the remainder of this paper, we give the models for application and architecture of the system in Sect. 2.2. The problem is formulated in Sect. 2.3. We give an introduction to control theory in Sect. 2.4. In Sect. 2.5, we present the details of our proposed optimization strategy with an illustrative example. Our optimization approach is evaluated in Sect. 2.6 on several test cases. The related work is covered in Sect. 2.7 and Sect. 2.8 concludes the paper.

2.2 System Model

This section presents the architecture and application models. Table.2.1 summarizes the notations used in the system model.

2.2.1 Architecture Model

There have been several FN architectures proposed, and some of them are commercially realized [PML^{+}19, HDNQ17, YLL15, FOR21, TTT21]. A possible FN architecture targeting mixed-criticality applications, is presented in Fig. 2.2. Such an architecture is similar to several FN architectures prepared for industrial applications [PML+19, TTT21]. We model the architecture as a set of FNs, denoted by $\mathcal{N}$. Each FN, $N_i \in \mathcal{N}$, has a set of cores $\mathcal{P}_i$, and each core is denoted with $P_j \in \mathcal{P}_i$. An example architecture with two FNs is presented in Fig. 2.3. The FNs have respectively two and one cores. Sensors and actuators are connected to FNs with network switches. The lines represent network links.

Mixed-criticality applications sharing the same platform have to be isolated from each other, otherwise a faulty lower-criticality task may interfere with a higher-criticality task, leading to failure. We assume that the applications are isolated from each other using spatial and temporal partitioning [Rus00], implemented via hypervisors such as ACRN [ACR20], Xen [The21], PikeOS [KW07] or XtratuM [Uni21].

Table 2.1: Summary of notation

Symbol	Fog Computing Platform (FCP)
$\mathcal{N}$	Set of all Fog Nodes
$N_i(\mathcal{P}_i) \in \mathcal{N}$	Fog Node (FN)
$\mathcal{P}_i$	Set of all cores in the Fog Node N_i
$P_j \in \mathcal{P}_i$	Core
$\mathfrak{P}$	Set of all cores in the platform
Δ	Set of all partitions
$\delta_i(L_i, \xi_i) \in \Delta$	Partition
L_i	Criticality level of a partition
ξ_i	Overhead time of a partition
$\mathcal{V}$	Set of partition tables
$v_i \in \mathcal{V}$	Partition table
$\mathcal{S}$	Set of schedule tables
$s_i \in \mathcal{S}$	Schedule table
Γ	Set of all applications
$\gamma_i \in \Gamma$	Set of tasks of application γ_i
Fi	Set of possible periods for an application
$\mathcal{F}$	The application mapping function to periods
$\tau_i(D_i, T_i, L_i, \mathcal{C}_i) \in \gamma_j$	Task
D_i	Deadline of a task
T_i	Period of a task
L_i	Criticality level of the application γ_j
$\mathcal{C}_i$	Set of WCETs for the application γ_j
$\mathcal{M}$	The task mapping function to the cores
$\mathcal{O}$	The task assignment function to the partitions

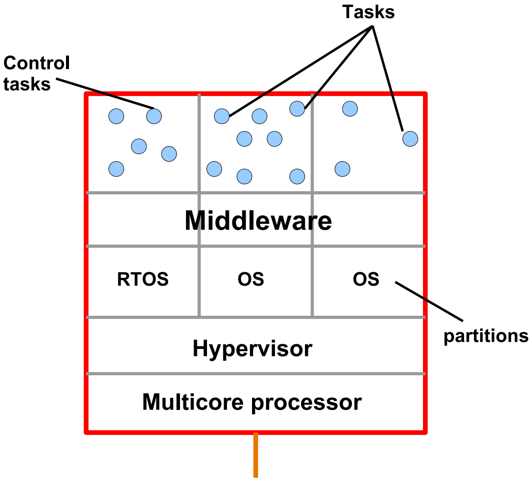


Figure 2.2: Fog node architecture. The software stack is running on a multicore, and has, from bottom to top, a hypervisor, partitions running OSes, middleware and tasks.

We denote the set of partitions with Δ. Each partition $\delta_i \in \Delta$, is characterized by a criticality level L_i. For example L_i can represent the Safety Integrity Level (SIL) of an application, which has values from 0, non-critical, to 4, highest criticality [Sto96]. A partition δ_i, scheduled on multiple cores, consists of several partition slices that are time slots to which the processor is assigned for the partition. We assume that the partitions are statically scheduled via partition tables, denoted with $\mathcal{V}$, (e.g, as used in Xen or PikeOS), which allocate processing cores from an FN to partitions in partition slices. A partition table repeats periodically with a system cycle. Switching among partition slices imposes an overhead. This overhead depends on the computing platform, the hypervisor, see [XXL$^+$14] for a description of overheads in Xen, and may also depend on the contents of the partition. For example in [ZSEP19], researchers assume that the overhead is 5% of the maximumWorst-Case Execution Time (WCET) of the tasks allocated to a partition. Our model is general, and assumes a partition-dependent overhead denoted with ξ_i for each partition δ_i.

Real-time applications can be implemented with time-triggered or event-triggered scheduling policies. In this paper, we assume that the scheduling policy is static cyclic scheduling [But11] (also known as time-triggered scheduling), which has been shown to be suitable for critical control applications. We will consider event-triggered scheduling in our future work. The set of all schedule tables in the model are denoted with $\mathcal{S}$. A schedule table $s_i \in \mathcal{S}$, captures the start and finishing time of tasks. We consider that within a schedule table, a task may be split into several parts, similar to run-time preemption in preemptive scheduling, but decided at design time. This has been shown

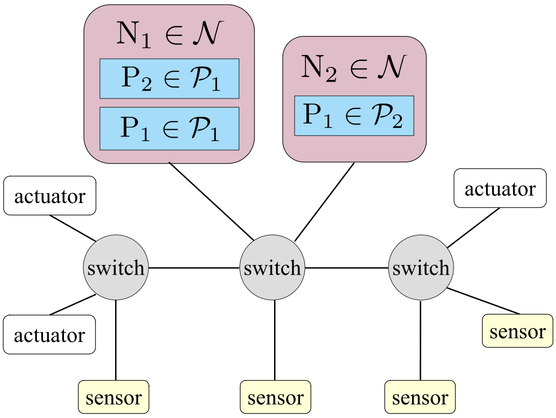


Figure 2.3: Example architecture with two FNs.

to improve flexibility, schedulability [Ves07] and QoC for control tasks [BCP19]. The preemption threshold can be controlled by a parameter called macrotick, which specifies the granularity of preemption [CSE14].

Fig. 2.4 shows an example of partition tables $\mathcal{V}$ and schedule tables $\mathcal{S}$ using a Gantt chart. In this example, we assume that mixed-criticality applications with a total of ten tasks are executing on two cores, which have four partitions Δ. All tasks in a partition have the same criticality level. The partitions δ_1, δ_2, δ_3, and δ_4 have the criticality level of respectively $L_1 = 1$, $L_2 = 0$, $L_3 = 3$, and $L_4 = 2$. The black lines represent the overhead times of the partitions. The task scheduling is depicted with white rectangles and the partition scheduling is depicted in colored rectangles.

2.2.2 Application Model

The set of all applications is denoted with Γ. An application is denoted with $\gamma_i \in \Gamma$ and composed of tasks $\tau_j \in \gamma_i$. Tasks may have data dependencies, which are modeled using a Directed Acyclic Graph (DAG), where nodes are tasks and edges represent data flows between the tasks. A data-dependent task is ready when all of its inputs have arrived. A task produces its outputs when it terminates. For example, as will be discussed in Sect.2.4.1, each control application is implemented as three data-dependent control tasks: a sampling task, a task that implements the control algorithm and an actuator task. Each task τ_i is periodic and has a period T_i, and a deadline D_i. The deadline is relative to the activation of the task. For each task τ_i, we know the set of WCETs $\mathcal{C}_i$ on the cores, where it is considered for mapping. The WCETs may be impacted by shared resources in a multicore, i.e., bus, memory, I/O. However, the problem of contention-

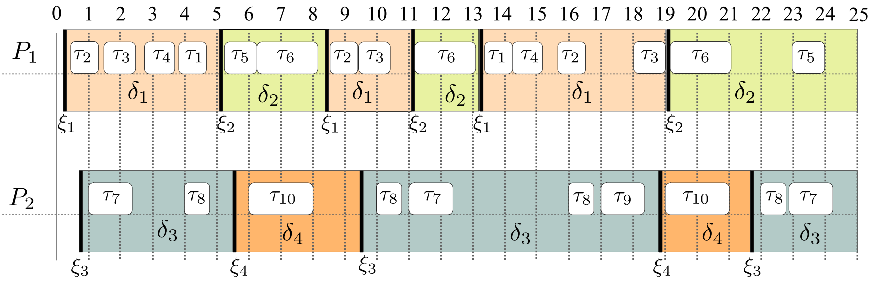

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
P_1
τ_2 τ_3 τ_4 τ_1 τ_5 τ_6 τ_2 τ_3 τ_6 τ_1 τ_4 τ_2 τ_3 τ_6 τ_5
δ_1 δ_2 δ_1 δ_2 δ_1 δ_2
ξ_1 ξ_2 ξ_1 ξ_2 ξ_1 ξ_2
P_2
τ_7 τ_8 τ_{10} τ_8 τ_7 τ_8 τ_9 τ_{10} τ_8 τ_7
δ_3 δ_4 δ_3 δ_4 δ_3
ξ_3 ξ_4 ξ_3 ξ_4 ξ_3

Figure 2.4: Example partition tables and schedule tables: Ten tasks are scheduled on two cores and mapped to four partitions.

aware scheduling is orthogonal to our work and we can use the techniques mentioned in [ZSB$^+$12] to account for the contention.

The mapping of tasks to the cores is modeled by using the function $\mathcal{M} : \tau_i \longrightarrow \mathfrak{P}$, where $\mathfrak{P}$ is the set of all cores in the platform. The system engineers may place constraints on the mapping of tasks, which can be handled by our model. The tasks are also assigned to partitions for execution. The assignment of the tasks to the the partitions is denoted by $\mathcal{O} : \tau_i \rightarrow \Delta$, where Δ is the set of all partitions in the system. The criticality level of an application γ_i is captured by its SIL L_i, see Sect. 2.2.1. Tasks can be assigned only to partitions that have the same criticality level.

We assume that tasks which have data dependencies share the same period. For a control application γ_i, we are given a set of possible periods F_j. We use the function $\mathcal{F}$ to capture the period $T_i = \mathcal{F}(\gamma_i) \in F_i$ of a control application γ_i. Our optimization strategy will select the period of the control application.

We show in Fig. 2.5 an example application model consisting of three applications. The criticality level, deadline and period of each task are depicted in the figure. The WCET of each task is also given considering a given mapping to a core. The values for deadlines, periods, and WCETs are in milliseconds. The application γ_2 is a control application with three tasks with precedence constraints. The application γ_2 has a set of possible periods F_2.

2.3 Problem Formulation

We formulate the problem as follows: Given (i) a set of applications Γ and (ii) a set of FNs $\mathcal{N}$, we want to determine a configuration Ψ consisting of: (1) a set of partitions Δ, (2) a mapping $\mathcal{M}$ of the tasks to cores, (3) an assignment $\mathcal{O}$ of tasks to the partitions,

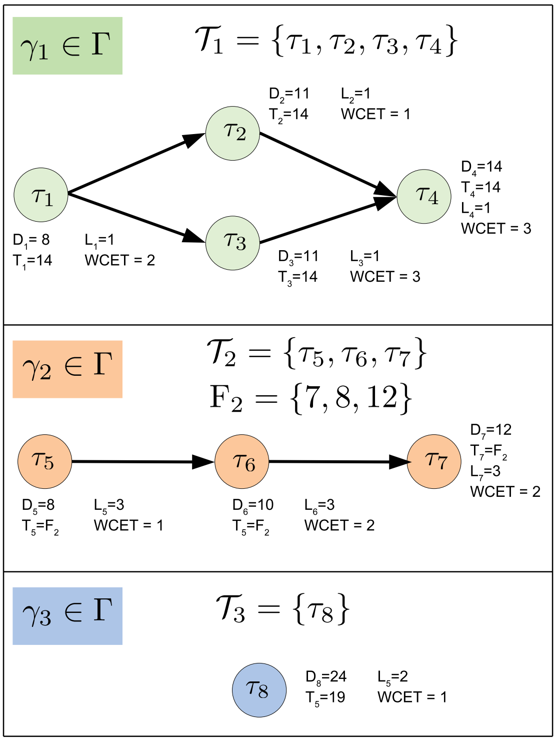


Figure 2.5: Example application model with three applications.

(4) the periods of control applications $\mathcal{F}$, (5) the partition tables $\mathcal{V}$, and (6) the schedule tables $\mathcal{S}$ such that:

1. **Maximum control performance is achieved for the critical control applications**: We seek a solution which has the best overall QoC for all the control applications. This is realized by minimizing the function $\bar{J}$ captured by Eq. (2.7), see Sect. 2.5.3.

2. **The deviation among the QoC of control applications is minimized**: We would like to balance the deviation σ_J, captured by Eq. (2.8), see Sect. 2.5.3.

3. **Temporal isolation is achieved among tasks with different criticality levels**: Each task τ_i and its assigned partition δ_j, captured with the function $\mathcal{O}$, share the same criticality level.

4. **The deadlines for all tasks are met**: Given that all the tasks are periodic and real-time, each task τ_i should completed before its deadline D_i.

2.4 Control Theory

The mathematical relation between the inputs, outputs and state variables of a dynamical system around an equilibrium point can be modeled as a linear differential equation and denoted by a state-space representation [OY02]

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) + w(t), \\ y(t) &= Cx(t) + Du(t), \end{aligned} \tag{2.1}$$

where the vectors x, u, w and y denote the state, the control input, the disturbance input, and the measured output respectively, and where A, B, C and D are matrices of appropriate sizes. The input–output relationship can equivalently be described by a transfer function $G(s)$ [OY02]. A Feedback Control System (FCS), or simply a control application, samples the output of the dynamical system $y(t)$, calculates the deviation from the desired output $r(t)$ (in this paper generically assumed to be zero), and drives the deviation to zero by applying an appropriate control signal $u(t)$.

2.4.1 Feedback Control System

An FCS can be implemented as a three task application. The source task, let's call it τ_1, samples the dynamical system by using sensors. The task may process the captured data from sensors. The second task, let's call it τ_2, uses the output of the task τ_1 to calculate the control signal. The task τ_2 utilizes various methods for the calculation and may be engaged with time-consuming calculation [OY02]. The implemented method for calculating the control signal is called the control law. The sink task, let's call it τ_3, uses the output of the task τ_2 to exert the control signal using the connection with the actuators. A simple FCS has an analogue to digital converter for the source task τ_1, a control law τ_2, and an analogue to digital converter for the sink task τ_3. Fig. 2.6 shows a simple FCS.

A control application is typically a periodic application with a known period. The period should be chosen in relation to the speed of the controlled system, and the shorter the period, the faster the controller is able to respond to the typical disturbances. On the other hand, a too short period causes high utilization of resources and leads to problems in resource-constrained computing platforms. A common rule of thumb [AW97] is to determine the period of the application based on the bandwidth of the closed-loop system. The closed-loop transfer function $H(s)$ is calculated by

$$H(s) = \frac{G(s)K(s)}{1 + G(s)K(s)}, \tag{2.2}$$

where $G(s)$ and $K(s)$ are the transfer functions of the dynamical system and the feedback controller respectively [OY02]. The sampling period T is then chosen in the

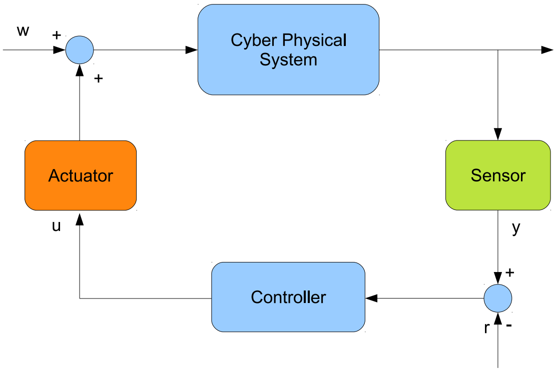


Figure 2.6: A simple FCS.

interval

$$\frac{0.2}{\omega_b} \leq T \leq \frac{0.6}{\omega_b}, \tag{2.3}$$

where ω_b is the 3 dB bandwidth of $H(s)$ [AW97].

As discussed, choosing the period from the interval has two impacts; first, stability and robustness of controller and last, resource utilization and schedulability. Our optimization strategy will determine the periods $\mathcal{F}$ to strike a compromise between their impacts.

The task timing is a source of additional disturbances for a control application. Ideally, the controller should execute without timing variations (jitter) and with as short delay as possible between the sensor task and the actuator task. A time delay has the direct consequence of decreasing the phase margin of the control system, which means worse performance and less robustness. Jitter is the deviation from the true periodic timing of an event, and its effects on the control performance are less obvious to analyze. In a control application, the event can be the execution of a task or the receiving of a network message.

The execution of a task is a periodic event of which instances are characterized by start time, duration and end time. Jitters can be associated with the start time, the end time and the duration. It is also associated either among all instances or two consecutive instances. The data packets to/from actuators/sensors are also periodic events which are characterized by send-time, transmit-time and receive-time which are vulnerable to jitter. While we are ignoring the communication in this paper, delays and jitter are only applied to tasks. The jitter of a task is either measured among all the instances (absolute) or two consecutive instances (relative). We categorize jitters as follows:

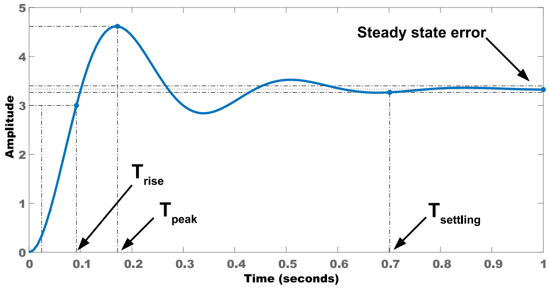


Figure 2.7: Step response of a sample control loop.

- **Start Jitter** of a task is the maximum deviation of the arrivals of instances of a task.
- **Release Jitter** of a task is the maximum deviation of the worst-case delay between the arrivals of instances of a task and their release times.
- **End Jitter** of a task is the maximum deviation of the release time of instances of a task.
- **Input–Output Jitter** is the maximum deviation of the worst-case delay between sampling from a cyber-physical system and exerting the actuation to it among the instances of tasks in a control loop. This type of jitter covers both timing of communication links form and to sensors and actuators, and the execution of the control tasks.

2.4.2 Control Design

While designing an FCS, there is a trade-off between accuracy and rapidity of the control loop. The trade-off is called the control performance. It is determined by several parameters such as the damping ratio, the phase margin and the gain margin, see [OY02] for more details. These parameters help control engineers to find the suitable control law and tune the control law to get the intended performance. The accuracy and rapidity is depicted in the transient and steady state response of the control loop. Fig. 2.7 shows the transient and steady state step-response of a sample control loop with notation of associated parameters (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 out-put 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.

The various aspects of the control loop performance can be captured in a cost function. A common choice [SLSS96] is to use a quadratic cost function of the form

$$J = \int_0^\infty \left(x^T(t)Q_1x(t) + u^T(t)Q_2u(t)\right) dt, \tag{2.4}$$

where the weighting matrices Q_1 and Q_2 tell how much deviations in the different states and the control input should be penalized. By proper tuning of the cost function, the desired transient and steady-state behaviour can be achieved in the control design. The same cost function can also be used to evaluate the performance of the controller under non-ideal circumstances. A larger value of the cost J then means that the response is more sluggish or more oscillatory, typically increasing the settling time of the system.

Given a linear system description by Eq. (2.1) and a quadratic cost function in Eq. (2.4), an optimal controller known as a linear-quadratic-Gaussian (LQG) controller can be calculated [OY02]. The above formulation is given in continuous time, but the LQG design methodology can also handle a large number of other conditions, such as sampled design [AW97] and compensation for time delays [Krs09]. The Jitterbug toolbox [LC02], utilized in this paper to design control applications, has support for designing an optimal sampled LQG controller that compensates for either a fixed or a random input–output delay with a given probability distribution.

2.4.3 Calculation of Control Performance

In this paper, we use JitterTime [CPBM19] to calculate the QoC with the cost function J, defined in Eq. (2.4). JitterTime takes the schedule tables $\mathcal{S}$ and partition tables $\mathcal{V}$ and calculates the cost J. The tables $\mathcal{S}$ contain the starting and finishing time of the tasks. JitterTime simulates the behaviour of a control application with the given starting and finishing times of control tasks and evaluates the behaviour using the quadratic cost function in Eq. (2.4).

The cost J decreases under the circumstances in which the Input-Output Jitter of a control application (defined in Sect. 2.4.1) as well as the end-to-end response of the control application decreases. The end-to-end response of a control application is the delay between the sampling from a cyber-physical system and exerting the actuation to it. In our problem, the delay is between the starting of the sensor task to the finishing of the actuator task. More information about the inner workings of JitterTime can be found in [CPBM19].

2.5 Solution

The problem we are addressing in this paper is interactable. Finding a solution to our problem involves deciding on the schedule tables, which has been shown to be Non-deterministic Polynomial time (NP)–complete in the strong sense [Ull75]. For such problems, exact optimization methods such as Brunch & Bound, Integer Linear Programming and Constraint Programming have exponential efforts. Hence, we propose a Simulated Annealing (SA)-based metaheuristic [BK05] to solve this optimization problem. Metaheuristics do not guarantee finding the optimal solution, but have been shown to find good quality solutions for a wide range of practical applications [BK05].

We have decided to divide the problem such that the schedule synthesis is performed separately within the SA using a scheduling heuristic. List Scheduling [Sin07] is a typical heuristic that derives good quality solutions, but it cannot easily handle applications with multiple periods and preemption. Instead, inspired by [CSE14], we have proposed a scheduling heuristic extended from [MPC19] based on the simulation of an Earliest Deadline First (EDF) algorithm, which can handle both multiple periods and preemption.

An overview of our proposed Fog Computing Platform Configuration (FCPC) optimization strategy in shown in Alg.1. The SA decides the period of the control applications, the mapping of tasks to cores in the FCP. The assignment of tasks to partitions and the partition and schedule tables are decided by our EDF-based Scheduling and Partitioning Heuristics (SPH, called in Alg.1). SA also decides parameters that influence the scheduling in SPH, such as task offsets Θ and relative deadlines Φ used for the EDF simulation.

SA is presented in Sect. 2.5.1 and SPH in Sect. 2.5.2. The objective function used for the optimization is presented in Sect. 2.5.3. SA uses design transformation to explore the search space, and these are presented in Sect. 2.5.4. Sect. 2.5.5 has an example that illustrates how our proposed FCPC strategy works.

2.5.1 Simulated Annealing

SA (line 7–17 in Alg. 1) starts from an initial solution (line 4) and iterates to search the solution space (line 7–17). The initial solution assigns the period of each control task to the minimum value in its set of periods $\mathcal{F}$, assigns the offsets Θ of tasks to zero, and sets the relative deadline Φ of all tasks to their deadline values. The initial mapping $\mathcal{M}$ is obtained by a greedy approach i.e., each task is mapped iteratively to the core that has the smaller utilization in that iteration. The initial assignment of tasks to partitions $\mathcal{O}$ is defined such that each application has a partition for its criticality level on each core where the application has a task mapped. The partition tables $\mathcal{V}$ and schedule tables $\mathcal{S}$ are obtained with our Scheduling and Partitioning Heuristic (SPH), called inside the InitialSolution function.

Algorithm 1 $\Psi =< \mathcal{M},\mathcal{O},\mathcal{S},\mathcal{V},\mathcal{F} >= FCPC(\Gamma,\mathcal{N})$

```
 1: $i \leftarrow 0$
 2: $t \leftarrow T_{start}$
 3: $\Theta \leftarrow \{0\};\Phi \leftarrow \{D_i\}$
 4: $\Psi \leftarrow$ InitialSolution$(\Gamma,\mathcal{N})$
 5: $J \leftarrow JitterTime(\mathcal{S},\mathcal{V},\Gamma)$
 6: $\Omega \leftarrow$ CostFunction$(J,\mathcal{S})$
 7: repeat
 8:     $< \mathcal{M}_i,,\mathcal{F}_i,\Theta_i,\Phi_i >\leftarrow$ Neighbor$(\Psi,\Gamma,\mathcal{N})$
 9:     $< \mathcal{S}_i,\mathcal{V}_i >\leftarrow$ SPH$(\mathcal{M}_i,\mathcal{F}_i,\Theta_i,\Phi_i,\Gamma,\mathcal{N})$
10:     $J_i \leftarrow JitterTime(\mathcal{S}_i,\mathcal{V}_i,\Gamma)$
11:     $\Omega_i \leftarrow$ CostFunction$(J_i,\mathcal{S}_i)$
12:     $\lambda \leftarrow \Omega_i - \Omega$
13:     if $\lambda < 0$ or $random[0,1) < Prob(\lambda,t)$ then
14:         $\Psi \leftarrow \Psi_i;\Theta \leftarrow \Theta_i;\Phi \leftarrow \Phi_i$
15:     end if
16:     $t \leftarrow t \times \alpha$
17: until stopping criterion is True
18: return $\Psi =< \mathcal{M},\mathcal{O},\mathcal{S},\mathcal{V},\mathcal{F} >$
```

In each iteration, SA uses design transformations (or moves) to generate neighboring solutions starting from the current solution Ψ (line 8). The generated neighborhood is evaluated with the cost function Ω, defined in Sect. 2.5.3. In each iteration, the algorithm compares the cost Ω_i of the generated neighborhood with the cost Ω of the current solution (line 13).

SA accepts a solution if the cost is improved. SA may also accept a worse-quality solution (in the hope to better explore the solution space) with a certain probability:

$$Prob(\lambda,t) = e^{-\frac{\lambda}{t}}, \tag{2.5}$$

where λ is the difference between cost of the generated neighborhood and cost of the current solution (line 12). The probability to accept worse solutions decreases with time according to a ”cooling schedule”, where t is the current temperature. SA starts from an initial temperature T_{start} (line 2), and cools down in each iteration at the rate of α (line 16). The search terminates when a stopping criterion has been satisfied (line 17), e.g., no improvement after a given number of iterations, a temperature of zero or a time limit was reached.

2.5.2 Scheduling and Partitioning Heuristic (SPH)

Our proposed Scheduling and Partitioning Heuristic (SPH) is presented in Alg.2 and takes as input a mapping $\mathcal{M}$, a set of periods $\mathcal{F}$, a set offsets Θ, a set of relative-deadlines Φ, the set of applications Γ, and the set of FNs $\mathcal{N}$. The main idea of SPH is to create first a schedule table for the tasks considering the mapping fixed by SA, and then to post-process the schedule table to derive the partitions and the allocation of tasks to the partition slices. Thus, SPH has two parts, the first part schedules the tasks (line 2–10) and the second part (line 11–13) groups the tasks together to form partitions. During the construction of the schedule table in the first phase, the SPH does not consider the partitioning.

As mentioned earlier, to derive the schedule tables $\mathcal{S}$, we perform at design time an EDF simulation. The output of that simulation is the set of schedules $\mathcal{S}$. In the simulation, we consider that the duration of each task is its WCET. With EDF, a task has the highest priority (and will be scheduled on its respective core) if its deadline D_i comes earlier considering the current time. The outcome S of a simulation is controlled by Θ, the tasks offsets (their initial earliest activation) and Φ, the relative deadlines used for each task in the simulation. These are modified for each task τ_i by our SA in Alg. 1, in the ranges D_i to T_i for the offsets Θ and 0 to D_i for the relative deadlines Φ. Our EDF simulation allows preemption (a higher priority task that is ready for execution may interrupt a lower priority task) considering the given macrotick, and can handle data dependencies, i.e. a task will not start before its predecessors have finished executing.

The EDF simulation is performed for the duration of a hyperperiod H (which is also the system cycle), defined as the Least Common Multiple of all the task periods (line 1). SPH starts by creating a queue Q_{jobs} with jobs of the tasks in Γ that have to run during H (line 2). Note that in our implementation these jobs are created on the fly, based on events occurring during the simulation. These events are generated by our simulation at design time, as part of the simulation used to derive the schedules. The simulation is performed in lines 4–10. Because SPH is run in each iteration of SA, we have optimized its implementation for speed, efficiency, simulation events and skipping only to events that have relevance for building the schedule tables $\mathcal{S}$. In the following, we explain how the simulation works in principle.

The simulation takes those jobs ∂ form Q_{jobs} that are ready to execute at the time t

Algorithm 2 $< \mathcal{O},\mathcal{S},\mathcal{V} >= SPH(\mathcal{M},\mathcal{F},\Theta,\Phi,\Gamma,\mathcal{N})$

```
1:  $H \leftarrow$ HyperPeriod($\Gamma$)
2:  $Q_{jobs} \leftarrow$ CreateJobs($\Gamma, H$)
3:  $t \leftarrow 0$
4:  repeat
5:      for all $\partial$ in $Q_{jobs}$ ready at $t$ do
6:          $\partial_H \leftarrow$ GetHighestPriority($\partial$)
7:          $\mathcal{X} \leftarrow$ Schedule($\partial_H$)
8:      end for
9:      $t \leftarrow$ NextEvent($Q_{jobs}, t$)
10: until $t < H$
11: $\chi \leftarrow$ GroupTasks($\mathcal{X}, \Gamma$)
12: $\mathcal{S} \leftarrow$ GenerateScheduleTable($\chi$)
13: $\mathcal{V} \leftarrow$ GeneratePartitionTable($\chi$)
14: return $< \mathcal{O},\mathcal{S},\mathcal{V} >$
```

(line 5) and sorts them based on their priority (line 6). The job which has the earliest deadline and its precedent jobs are arrived, has the highest priority. The high-priority job is denoted with ∂_H. If the priority of ∂_H is higher than the currently executing job, SPH preempts it and schedule ∂_H instead (line 7). The simulation is stored in $\mathcal{X}$. SPH determines the next time in which a job becomes ready (line 9), considering the remainder of jobs in Q_{jobs} and the macrotick parameter mentioned in Sect. 2.2.1, which controls the granularity of preemption.

The final part of SPH post-processes the simulation data structure $\mathcal{X}$. SPH groups the time-wise consecutive jobs which have the same criticality level to form partitions, and also delays the tasks to insert the required task switching (in case preemptions were introduced) and partition overheads ξ_i (line 11). See Sect. 2.5.5 for an illustration on how our heuristic works to create partitions by grouping tasks. SPH extracts the schedule tables $\mathcal{S}$ from the simulation χ (line 12), and the partition tables $\mathcal{V}$ (line 13).

2.5.3 Cost Function

In this section, we define the weighted cost function Ω in Eq.(2.6), used by our FCPC optimization strategy. The function has three terms (QoC, deviation of QoC and task schedulability constraint, respectively) and takes the QoC of control applications J and the schedule tables $\mathcal{S}$ as input. The QoC optimization is controlled by the weights β_1 and β_2, whereas β_3 is a penalty value for the case when task deadlines are missed. The weights allow the system engineer to control the search for schedulable solutions that optimize QoC. Larger values for β_1 and β_2 will drive the search to optimize QoC, whereas a larger value for β_3 will drive the search faster to schedulable solutions.

$$\Omega = \beta_1 \times \bar{J} + \beta_2 \times \sigma_J + \beta_3 \times \Lambda \tag{2.6}$$

The control performance of control applications is captured by the first term. Assuming m number of control applications, the average QoC for the applications is

$$\bar{J} = \frac{\sum_{i=1}^{m} J_i}{m}, \tag{2.7}$$

where, J_i is the QoC for a control application γ_i which is calculated by JitterTime and its value is mapped to the range $[0,1]$. The range of cost performance is from 0, for the best-performance, to 1, for the worst-performance, e.g., which is unstable. $\bar{J}$ is normalized to the same range.

The second term captures the deviation among the QoC of the control applications, and is defined in Eq. (2.8). Concerning the range of J, the range of variation is from 0, for the equally distributed QoC of control applications, to $\sqrt{\frac{m-1}{m}} < 1$ for m control applications when their performance costs are highly-deviated.

$$\sigma_J = \sqrt{\frac{\sum_{i=1}^{m} |J_i - \bar{J}|}{m}} \tag{2.8}$$

The last term is the function Λ, which is a constraint that checks for deadline violations for all the tasks in the schedule table. Λ is also normalized, and starts form 0, for no deadline violations, to 1, for the case in which all the jobs have missed their deadlines.

2.5.4 SA Design Transformations

As mentioned earlier, SA decide the mapping $\mathcal{M}$ of tasks to the core, assignment $\mathcal{O}$ of tasks to partitions, and periods $\mathcal{F}$ of control applications. The SA also varies the offsets of tasks and EDF deadline to create different scenarios for the EDF simulation, generating various schedules. SA uses moves to explore the solution space by generating randomly neighborhoods of the current solution. SA randomly selects one of the moves and applies it to randomly selected tasks to generate the neighborhood in each iteration. The moves are:

- **Swap Tasks**: swaps the mapping of two selected tasks.

- **Period Selection**: randomly chooses a period T_i from the given set of periods F_i for the selected control task τ_i.
- **Deadline Adjustment**: randomly selects a relative deadline Φ (used in the EDF simulation in SPH) in the range from D_i to T_i.
- **Offset Adjustment**: the offset Θ of the selected task τ_i is selected randomly in the interval from 0 to D_i.

To drive the search faster towards the schedulable solution with increased QoC, we encourage SA to pick tasks that need special attention, e.g., because they missed their deadline or they impact QoC. This is achieved by sorting the tasks based on the criteria we want to address (schedulability, QoC) and selecting randomly tasks based on probability density function that are skewed towards the head of the sorted list.

2.5.5 Illustrative Example for FCPC

Let us present an example illustrating how FCPC works. We have two cores, P_1 and P_2 and four applications, including two control applications γ_1 and γ_4. The applications have 12 tasks in total. Each control application is controlling an inverted pendulum in the upright position, with its process modeled as

$$G(s) = \frac{200}{s^2 + 400}. \tag{2.9}$$

Each of the control applications has three tasks which are respectively sensor, LQG controller, and actuator task, see Sect. 2.4.1. The controller tasks, τ_2 in the control application γ_1, and τ_9 in the control application γ_4 are LQG controllers which are designed using Jitterbug [LC02]. Table 2.2 shows the applications, tasks and their details.

FCPC starts with an initial configuration which comes from the initial solution (Sect. 2.5.1, line 4 in Alg. 1). SPH uses this initial configuration and creates an EDF simulation (Alg. 2). We take the stored simulation $\mathcal{X}$ (line 7 in Alg. 2), group the task to create partitions (line 11 in Alg. 2) and generate a schedule table (line 12) and a partition table (line 13). Let us explain how partitions are create, starting from the schedule in Fig. 2.8a, which is the result of EDF simulation stored in $\mathcal{X}$. SPH post-processes the schedule from left to right, and, if two tasks share the same criticality levels, it group them into same partition. Let us call this configuration CONF/S (from configuration with separation) depicted in Fig. 2.8b, which shows a part of the schedule table starting from 0 ms to 40 ms. We use different colors to highlight the partitions in Fig. 2.8. For

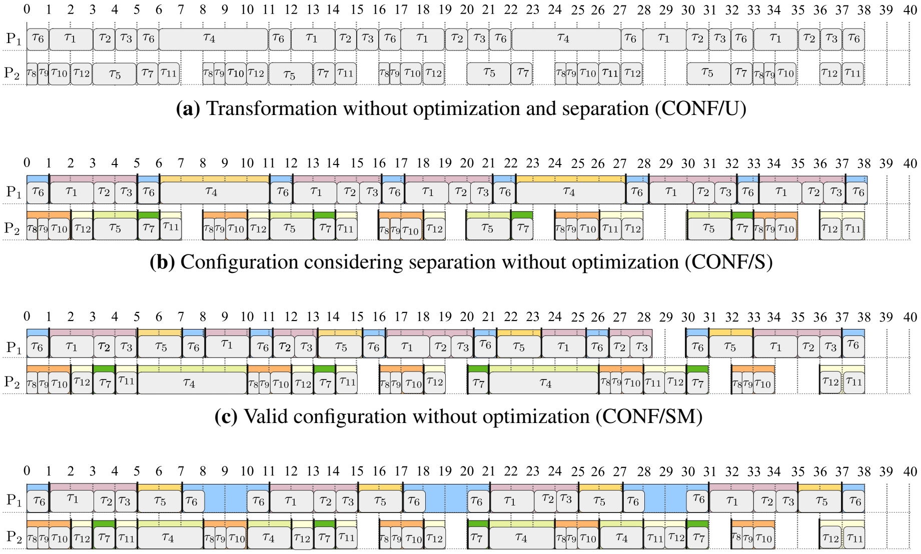
Table 2.2: Illustrative example applications.

Application	Tasks	L	WCET (ms)	T (ms)	D (ms)
γ_1	τ_1	3	2	F_1	12
	τ_2	3	1	F_1	12
$F_1 = \{6, 8, 10, 12\}$	τ_3	3	1	F_1	12
γ_2	τ_4	1	5	20	20
	τ_5	1	2	10	10
γ_3	τ_6	2	1	5	5
	τ_7	2	1	10	10
γ_4	τ_8	3	0.5	F_4	12
	τ_9	3	1	F_4	12
$F_4 = \{5, 8, 10, 12\}$	τ_{10}	3	0.5	F_4	12
γ_5	τ_{11}	0	1	12	12
	τ_{12}	0	1	9	9

example, we create four partitions, denoted with δ_4, δ_5, δ_6 and δ_7 on core P_2, with criticality levels of $L_{\delta_4} = 1$, $L_{\delta_5} = 2$, $L_{\delta_6} = 3$ and $L_{\delta_7} = 0$.

The overhead times for these partitions are determined as 100 μs, 50 μs, 100 μs and 50 μs, respectively (we use the approach from [ZSEP19], which considers the partition overheads of 5% of the largest task WCET in the partition). Regarding the task switching overheads, we use the values measured in [CSE14]. SPH may delay the tasks to apply these overheads, hence several instances of tasks may miss their deadlines. For example, the task τ_3 will miss its deadline at $t = 16$ ms for about 550 μs.

Let us consider that CONF/S is the current solution driving the search performed by SA in Alg. 1, and this SA perform a ”Swap Task” design transformation in line. 8, which results in swapping the mapping of tasks τ_4 and τ_5. This will result in the configuration from Fig. 2.8c, which we call CONF/SM (from configuration with separation and mapping), which is feasible, i.e., there are no deadline misses. As a consequence of the task swapping, not only mapping of the tasks to the cores are swapped but also their assignments to the partitions, since they have the same criticality level. In this configuration, the control application γ_1 experiences maximum I/O jitter (12.5% of its period, which is 8 ms) and the control application γ_4 has no I/O jitter (release and start jitters are seen at $t = 10$ ms and $t = 16$ ms). The cost of control is calculated by JitterTime, and the average of the two control applications is 0.09642 and the deviation is equal to 0.0388. The cost function, calculated as in Eq. (2.6) considering a value of 1 for all weights, has a value of 0.13522.



(a) Transformation without optimization and separation (CONF/U)

(b) Configuration considering separation without optimization (CONF/S)

(c) Valid configuration without optimization (CONF/SM)

(d) Optimized configuration (CONF/O)

Figure 2.8: Four different configuration for the applications in Table. 2.2; The black lines show the overheads of the partitions: CONF/U has no temporal separation; CONF/S has several deadline misses; CONF/SM is feasible and the cost function value is 0.13522; CONF/O is the final solution which shows 77% improvement.

SA will accept the configuration in Fig. 2.8c as the current solution, since it improves over Fig. 2.8b that had deadline misses (line 13 in Alg. 1). Let us assume that the next design transformation is done by ”Period Selection”, e.g., by selecting the period of 10 ms for tasks in the control application γ_1. The resulted optimized configuration (CONF/O) is depicted in Fig. 2.8d. The values of average QoC and the deviations (terms one and two in Eq. (2.6)) are 0.0268 and 0.0033, respectively, resulting in a cost function of 0.0301, which is an improvement of 77% over the CONF/SM in Fig. 2.8c

2.6 Experimental Evaluation

Our proposed optimization strategy, Fog Computing Platform Configuration (FCPC), was implemented in C#, and all the experiments were run on a laptop with an i7 CPU at 3.0 GHz and 32 GB of RAM. We investigate the performance of our proposed method on ten test cases, which have mixed-criticality tasks. The details of test cases are shown in Table 2.3, where column 2 shows the total number of cores in the FCP, column 3

Table 2.3: Evaluation results for our proposed optimization

Test cases	No. of Cores	Total No. of Control Applications	Total No. of Tasks	Total No. of Tasks for Criticality level of [0-4]	Ω of FCPC	Ω for FCPC/M	Ω for FCPC/Q	Ω for FCPC/P
1	2	2	12	{1,2,2,1,6}	0,19	48%	Not Feasible	60%
2	2	3	23	{3,2,5,4,9}	0,34	5%	Not Feasible	Not Feasible
3	2	2	17	{2,2,4,3,6}	0,21	5%	Not Feasible	33%
4	2	2	23	{2,1,7,6,7}	0,29	13%	Not Feasible	13%
5	3	3	32	{1,4,8,8,11}	0,21	39%	Not Feasible	55%
6	3	3	31	{2,3,9,7,10}	0,22	85%	Not Feasible	50%
7	3	4	33	{4,4,8,7,12}	0,26	15%	Not Feasible	10%
8	4	4	44	{4,6,11,9,14}	0,20	29%	Not Feasible	72%
9	5	6	54	{7,5,14,10,18}	0,21	21%	Not Feasible	Not Feasible
10	6	7	63	{6,7,16,12,22}	0,24	21%	Not Feasible	46%

shows the total number of control applications, column 4 shows the total number of tasks, and column 5 shows the total number of tasks having a particular criticality level, 0 to 4. Each test case has multiple control applications: Each control application has three control tasks (see Sect, 2.4.1) and the control task which implements the controller is a LQG controller designed with Jitterbug to control a plant using one of the three different processes which are defined in Eq. (2.9), Eq. (2.10), and Eq. (2.11),

$$G(s) = \frac{300}{s^2 - 200}, \tag{2.10}$$

$$G(s) = \frac{100}{s^2 + 300}. \tag{2.11}$$

The tasks in each test case represent real-time tasks with different criticality levels and can be run on any of the cores. Tasks with the same criticality level are mapped to the same partition and overheads are applied to each partition slice.

The results of evaluation are presented in Table 2.3. The Ω columns show the cost function of test cases for each solution. The results obtained by running FCPC on each test cases are reported in column 6 using the value of the cost function Ω. We have set the weights β_1, β_2 and β_3 to 0.45, 0.1 and 1.0, respectively. The weights were determined experimentally to guide the search faster towards solutions with optimized QoC. β_1 can be set by analyzing the stability of the control applications with Jitterbug and choosing a value that drives the search towards stable control. Jitterbug also reports the phase margins (smaller phase margin means larger sensitivity) of the applications and β_2 is set to allow a larger deviation from the mean QoC if there is a large variation among the phase margins of the applications. A β_3 value of 1.0 is a relatively large penalty value considering that the cost function terms are in the range $[0,1]$. To determine the ability of FCPC to improve the QoC measured by Ω, Table 2.3 also reports the results obtained by three variants of FCPC, as follows:

- **FCPC/M**: does not optimize the mapping of tasks and uses the mapping determined in the initial solution, as explained in Sect. 2.5.4.
- **FCPC/Q**: does all the optimizations of FCPC but does not use the QoC in the cost function (the first two terms), hence it optimizes only for schedulablity, ignoring the control performance.
- **FCPC/P**: generates solutions with our proposed strategy without considering period selection for critical control tasks. The period of control tasks are set to their smallest value.

The other Ω columns show the value in terms of percentage in deterioration of the cost function for FCPC/M, FCPC/Q and FCPC/P, respectively, compared to FCPC. A larger cost function value, i.e., larger percentage deterioration, means a worse-quality solution.

As we can see from Table 2.3, FCPC has been able to obtain feasible solutions for all the test cases, i.e., all the tasks' deadlines are satisfied and all the controllers are stable and have good QoC. The average value of cost function Ω is 0.24 and the values are not highly-deviated in all the test cases. We have used a time limit of 20 to 70 minutes as a termination criteria for FCPC and its variants, depending on the size of the test case.

When comparing FCPC with its variants that ignore certain optimization aspects, we can see in the last three columns of Table 2.3 large percentage deterioration, or even "unstable" control applications (shown with "Not Feasible") for nearly all the test cases. For example, the results show that not considering QoC in FCPC/Q gives the worst results, which demonstrates that the JitterTime-based QoC evaluation of solutions needs to be used during the optimization of a FCP configuration. Otherwise, the controllers become unstable even though the deadlines are not missed. Determining the right period for the control applications is very important as we can see in case of FCPC/P, where considering a single period results in control applications that are unstable and the degradation is high, on average 42% in all other cases. The results also show the importance of mapping, since using in FCPC/M, the mapping determined by the initial solution, the degradation is on the average 28%.

2.6.1 Realistic Test Case

We have evaluated the proposed optimization strategy on a realistic test case which consists of 8 applications running on a fog node inside a vehicle. Future vehicles are envisioned to be "fog nodes on wheels" [CBB17] as they integrate more and more functions and become interconnected with each other.

The details of the test case are in Table 2.4. We have 8 applications running on a dual-core fog node which include a drive-assistance application for radar-cruise control (application γ_4). The car is modeled with a first-order transfer function and the controller is a LQG speed controller which is design by Jitterbug. Application γ_1 monitors the engine, γ_2 is a passenger comfort application that controls the climate, γ_3 is used for image analysis as part of driver-assistance functionality. We give the applications different critically levels, based on their importance, as presented in the table.

Our proposed optimization strategy has successfully scheduled all the tasks and decided the task mapping to the partitions and cores. The results show that none of the tasks

Table 2.4: Realistic test case

Application	Tasks	L	WCET (ms)	T (ms)	D (ms)
γ_1	τ_1	2	0.5	10	10
	τ_2	2	0.5	10	10
	τ_3	2	1	10	10
γ_2	τ_4	1	2	15	15
	τ_5	1	2	15	15
	τ_6	1	1.5	15	15
	τ_7	1	1	15	15
	τ_8	1	2	15	15
γ_3	τ_9	2	1	20	20
	τ_{10}	2	0.5	20	20
	τ_{11}	2	0.5	20	20
	τ_{12}	2	1	20	20
	τ_{13}	2	1.5	20	20
	τ_{14}	2	0.5	20	20
γ_4	τ_{15}	3	0.5	F_4	24
	τ_{16}	3	2.5	F_4	24
	τ_{17}	3	3	F_4	24
	τ_{18}	3	1	F_4	24
	τ_{19}	3	3	F_4	24
	τ_{20}	3	1	F_4	24
$F_4 = \{18,20,22,24\}$	τ_{21}	3	0.5	F_4	24

has missed its deadline. Furthermore, the mapping of tasks to the cores shows the core utilization of 86.67% and 86.88% for the dual-core processor. We used JitterTime to simulate the controller behavior and calculate the cost of control for the application concerning the given cost function in 2.4.3 with the weights of 0.25, and 0.25 and 1 for β_1, β_2 and β_3. The cost function has the value of 0.007.

2.7 Related Work

There is already much work on various topics related to Fog Computing [YLH$^+$18, MKB18, MNY$^+$18]. Even though basic quality-of-service (*QoS*) for applications has been addressed, the QoC for control applications in the fog is still an open issue. However, there is a lot of useful literature in works that tackle the problem of degradation of control applications [ZHY16, ZSWY17, JX07].

Researchers propose several approaches (such as partial and spatial separation of control tasks, virtualization of PLCs, scheduling of control tasks, co-design of control applications) that guarantee extra-functional properties of control applications [SSS17, ZBP01, WS12, MH18]. The presented approaches are well studied and categorized into a category for platform configuration and a category for the integration of applications. Good performance for control applications will be ensured if both the applications and the platforms are configured.

Separation and isolation of applications regarding criticality levels ensure resource allocation to control applications. Researchers propose spatial and temporal separation to integrate mixed-criticality applications [WS12, MH18, JX07]. In this approach, partitions with different criticality levels separate the application. Resources are allocated to the partitions based on the criticality level, which assures resource availability and accessibility for critical applications with high priority. These applications would have guaranteed dependability by promising separation. A presentation of scheduling in mixed-criticality systems, which also considers partitioning, is presented in [BD13]. For example, Tamas-Selicean et al. [TSP15a] propose a method in which different SILs are assigned to the applications. In this method, applications with the same SIL are mapped to a single partition. Each partition is allocated several time slots on a processor to execute respective tasks. Concerning this method, the approach can provide a partition for each control application.

On the other hand, integration of the applications in the platform affects their functional and extra-functional properties. The co-design of control applications configure them at integration level to achieve the highest performance. The co-design approach takes the platform characteristics into account while designing the application to have good integration with the platform.

The QoC analysis and schedulability of the tasks are taken into account while designing the control applications in [ÅCES00] and [XCÅ16]. In the proposed approaches, the task scheduler schedules the tasks concerning the QoC of control applications. Besides, co-design is used to determine the period of tasks and design a robust and optimal controller. Other researchers also focused on the co-design and scheduling of control tasks to achieve the maximum QoC for control applications [WS12, MH18, JX07]. Co-design and scheduling concerning QoC for control applications are also proposed in the seminal work of Seto et al [SLSS96]. The authors optimize the period of control applications concerning the QoC and schedulability of the tasks.

Chwa et al. [CSL18] propose a co-design and scheduling method to maximize QoC for control applications. The authors assign a sampling period, and a maximum number of consecutive deadline misses as parameters for each task concerning system stability. Then, the tasks are scheduled concerning the parameters without compromising system stability and also with efficient use of resources. Mahmoud et al. [MH18] use optimization algorithms to derive a timing constraint of control tasks such as the task period to achieve maximum QoC for the control applications concerning the schedulability of the control tasks [RHS97]. In this work, heuristic algorithms are used to derive the period of tasks, deadlines of the tasks and end-to-end response of the control loop. The assigned parameters are assessed in a simulation that schedules and executes the tasks.

Samii et al. [SCEP09] present an approach in which a controller is synthesized for each plant, and the control tasks are scheduled concerning the priority of the tasks. In this work, the scheduling is based on the cost of control function, which aims to consider the maximum QoC for all the control applications. The same approach concerning co-design and scheduling of control application is used in [MSZ11]. A similar co-design approach is presented by Cervin et al. [CEBÅ02]. In this work, the scheduler uses feedback from execution time and also feed-forwards the workload along with the cost of control to achieve the best QoC. Besides, control task parameters such as period are changed with the feedback from execution time. The approach can compensate the impact of jitter on the QoC.

Task scheduling has a significant impact on the performance of control applications. The QoC-aware scheduling reduces the degradation of control applications to some levels based on the criticality of the application. Configuration of the platform at the computation level, especially in the task scheduler guarantees good performance for control applications. Barzegaran et al. [BCP19] have presented a heuristic approach for scheduling of tasks and mapping them to the cores which maximizes the QoC of control applications. The work also shows that allowing preemption in scheduling of tasks improves the schedulability of tasks and QoC of control applications. The work does not consider separation of mixed-criticality tasks which is covered in this paper, and it also ignores the effect of control tasks periods on the schedulability, which is also covered in this paper. Task period selection and cost function definition for the optimal control behavior is based on the cost of the control. The scheduling aims to cover the bounded jitter and latency regarding the stability margin of the control applications. In the work by Schneider et al. [SGMC12], the QoC measurement is embedded in the task scheduler with allowed preemption. The scheduler is capable of handling mixed-criticality applications as well.

Another co-design approach is considered in [SGM$^+$16]. The authors get feedback of delay and jitter in the execution of tasks from the scheduler and feed it to the control applications. The controller takes the feedback and adjusts the control output to compensate the delay and jitter impact and to maximize the QoC. The feedback from the task scheduler is also used to predict the jitter and delay. In [TG11], a feedback scheduling framework is developed to schedule control tasks such that the QoC is maximized for control applications and to adjust workload constraints. The QoC measurement is embedded in the task scheduler. The scheduler gets delay and jitter feedback to change the period of the tasks concerning the QoC and workload management. The same approach is used in [SGAN$^+$16], to schedule tasks concerning the QoC with feedback from scheduling. In this work, the period of tasks is changed regarding the feedback. Eker et al. [EHÅ00], propose a similar method. The method uses feedback from the scheduler to assign the period of control tasks. The period assignment provides good control performance along with optimizing the resource allocation of the task.

Cha et al. [CJK16] propose a method for scheduling of control tasks which determines the deadline and period of the tasks for achieving maximum QoC. The method optimizes the QoC of the control applications and resource utilization. In co-design approach presented in [XCÅ18], the task scheduler guarantees bounded delay and jitter in execution of control application while the co-design approach guarantees that the control application is still stable in the presence of bounded delay and jitter. In work by Fan et al. [FQ12], a scheduling algorithm is proposed that can provide some degree of isolation, which can host control applications. In this work, control applications can be assigned to partitions to ensure the separation. The algorithm also maps the tasks to the cores.

2.8 Conclusions and Future Work

In this paper, we have addressed the problem of configuring the mapping, partitioning, scheduling and periods of mixed-criticality tasks when implementing the applications on a Fog Computing Platform. The optimized solution has good and balanced Quality-of-Control for critical control applications, ensuring the schedulability of all real-time tasks, as well as spatial and temporal isolation for mixed-criticality tasks. Our proposed strategy is based-on a Simulated Annealing metaheuristics, which uses an Earliest Deadline First simulation.

We have evaluated this strategy on several test cases. As the results show, our proposed optimization strategy successfully generates solutions which have good and balanced Quality-of-Control for control applications considering temporal isolation for all the test cases in comparison with the solutions that have ignored some of the optimization criteria.

The successful virtualization of control, achieving the same control performance (and dependability) as the one taken for granted in OT, is a crucial step towards the adoption of Fog Computing in the industrial area. In our future work, we will consider the effect of the communication; we will take into account the possibility of incremental scheduling based on our proposed strategy, and we will also consider other optimization techniques such as constraint programming to solve the problem.

CHAPTER 3

Paper B: Communication Scheduling for Control Performance in TSN-based Fog Computing Platforms

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++.

3.1 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 Technology (OT) and Information Technology (IT), which are currently separated in a hierarchical pyramid (Purdue Reference Model [Wil94]) 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 Internet of Things (IIoT) [BMZA12], 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" [MG$^+$11]. 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" [Ope21a]. We define *Edge Computing* is as a new architectural paradigm in which the resources of an edge server are placed at the edge of the Internet, in close proximity to cyber-physical systems, mobile devices, sensors and Internet of Things (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 [BMNZ14]. 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 [IEE21b], upcoming 5G wireless standards [DMP$^+$14], and interoperability standards such as OPC Unified Architecture (OPC UA) [MLD09], all integrated into a Fog Computing Platform (FCP), see Fig. 3.1, which brings computation, communication and storage closer to the edge of the network. Several initiatives are currently working towards realizing this vision [PML$^+$19, HDNQ17].

The integration of computational and storage resources into the communication devices is realized in the Fog Node (FN), see Fig. 3.1. 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 [BMNZ14, PML$^+$19, PZB$^+$21] and FN solutions have started to be developed by companies [PML$^+$19, HDNQ17, TTT21].

Regarding the communication infrastructure, today, industry uses mostly proprietary protocols [GJF12] 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 [PRGS18], i.e., IEEE 802.1 TSN [IEE21b], see Fig. 3.1. 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 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 [IEE17], 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.

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

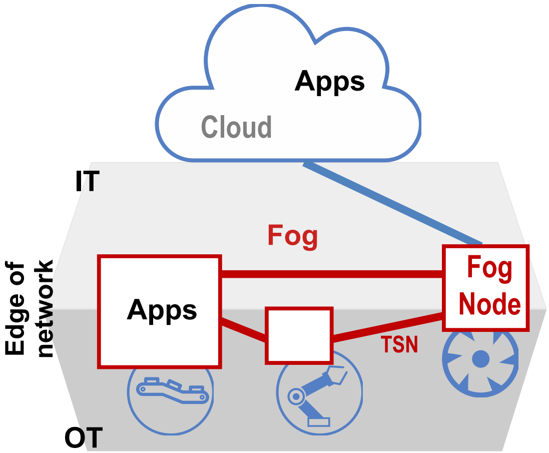


Figure 3.1: Fog Computing Platform: Boxes represent fog nodes, placed at the edge of network where OT and IT converge, connected to each other and to the Cloud in IT and to the industrial “thing" in OT, and running applications (Apps). Thick lines are physical TSN links.

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 [BZP20]. 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 [BCP20], 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.

3.1.1 Contributions

The related work, discussed in Sect. 3.7, 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 [MAS$^+$18, BZP20]. 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.

3.1.2 Outline of the Paper

The system model is presented in Sect. 3.2 where architecture, application and the internals of a TSN switch are described. We formulate our problem in Sect. 3.3. An introduction to control theory is presented in Sect. 3.4. In Sect. 3.5, the details of our proposed method are given. We evaluate our proposed method in Sect. 3.6 on several test cases. The related work in presented in Sect. 3.7 and Sect. 3.8 concludes the paper.

3.2 System Model

This section presents the architecture and application models. Table 3.1 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.

3.2.1 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 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 $\varepsilon_{i,j} \in \mathcal{E}$ and $\varepsilon_{j,i} \in \mathcal{E}$; a link is attached to one port of the node v_i and one port of the node v_j.

Each link $\varepsilon_{i,j}$ 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 $\varepsilon_{i,j}.d(size)$ is calculated based the frame's size, the physical medium, and the link length. The function d is a notation used in the constraints in Sect. 3.5 and it is attached to the link concept, i.e., $\varepsilon.d(size)$ means $d(\varepsilon, size)$.

Table 3.1: Summary of the notation

Notation	Definition
$\mathcal{G}$	Network Graph
ES	End-System
SW	Network switch
$v_i \in \mathcal{V}$	Network node
$v_i.d(c)$	Forwarding delay
$p_j \in v_i.P$	Egress port
$q_j \in p_i.Q$	Priority queue
$\varepsilon_{i,j} \in \mathcal{E}$	Link
$\varepsilon_{i,j}.s$	Link speed
$\varepsilon_{i,j}.d(c)$	Link propagation delay
$\varepsilon_{i,j}.mt$	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
$f_{i,m}^k$	Frame
$f_{i,m}^k.\phi$	Frame offset
$f_{i,m}^k.l$	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

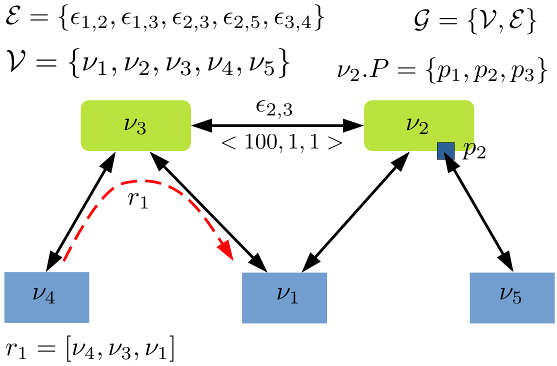

$\mathcal{E} = \{\epsilon_{1,2}, \epsilon_{1,3}, \epsilon_{2,3}, \epsilon_{2,5}, \epsilon_{3,4}\}$
$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$
$\mathcal{V} = \{\nu_1, \nu_2, \nu_3, \nu_4, \nu_5\}$
$\nu_2.P = \{p_1, p_2, p_3\}$
ν_3
$\epsilon_{2,3}$
$< 100, 1, 1 >$
ν_2
p_2
r_1
ν_4
ν_1
ν_5
$r_1 = [\nu_4, \nu_3, \nu_1]$

Figure 3.2: Architecture model example: the blue boxes are end systems, the green boxes are switches and the thick arrows are full-duplex physical links.

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 ESs two SWs is presented in Fig. 3.2, 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.

3.2.2 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 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. 3.3. 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 [IEE14]), 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,

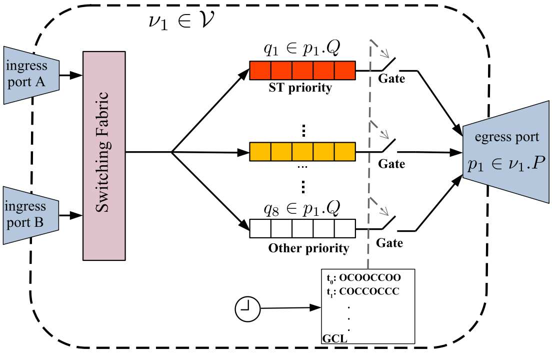


Figure 3.3: TSN switch internals.

similar to [COCS16]. 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 [COCS16] 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 [Cer03], 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 $v_i.d(c)$ 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 [SGJM$^+$20].

3.2.3 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 [BCP20] 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. 3.3

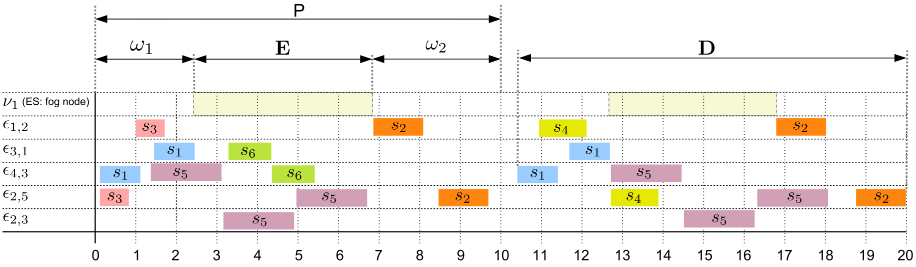


Figure 3.4: Example solution: ν_1 is a fog node and runs the control function during the execution slice denoted with **E**. The flow s_1 is the input flow and the flow s_2 is the output flow. The control application's period denoted with P, is 10 ms (s_1 and s_2 have the same period).

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 ESs, 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

Table 3.2: 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	$\langle \varepsilon_{4,3}, \varepsilon_{3,1} \rangle$
s_2	$\gamma_1.\mathcal{O}$[1]	0	240	10	10	$\langle \varepsilon_{1,2}, \varepsilon_{2,5} \rangle$
s_3	$\gamma_2.\mathcal{I}$[1]	1	90	20	20	$\langle \varepsilon_{5,2}, \varepsilon_{2,1} \rangle$
s_4	$\gamma_2.\mathcal{O}$[1]	1	180	20	20	$\langle \varepsilon_{1,2}, \varepsilon_{2,5} \rangle$
s_5	RT	0	450	12	12	$\langle \varepsilon_{4,3}.\varepsilon_{3,2}, \varepsilon_{2,5} \rangle$
s_6	RT	1	160	16	16	$\langle \varepsilon_{1,3}, \varepsilon_{3,4} \rangle$

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

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 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 [GZRP18] 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. 3.5.2 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 $f_{i,m}^k$. A frame $f_{i,m}^k$ 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. 3.4 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, ESs 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).

3.3 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. 3.4.3, 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. 3.2 and the flows from Table 3.2 is presented in Fig. 3.4. 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 [CO16] 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 [BCP20]. 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. 3.4, see see Sect. 3.5.3 for its definition.

3.4 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 System (FCS) in Sect. 3.4.1 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. 3.4.2. Finally, we define in Sect. 3.4.3 the QoC and present the approach we use in this work for calculating it.

3.4.1 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 [OY02]. The transfer function, commonly called *Plant*, is defined in the form of

$$Y(s) = G(s) \times X(s), \tag{3.1}$$

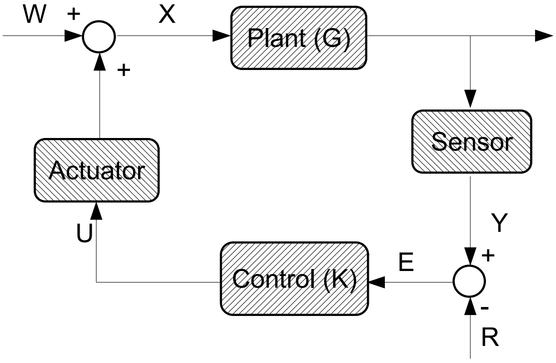


Figure 3.5: A simple FCS.

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. 3.5, 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 [AW97]. 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 [OY02].

3.4.2 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. 3.2.3 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 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. 3.5.2.

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 [OY02] 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. 3.6 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.

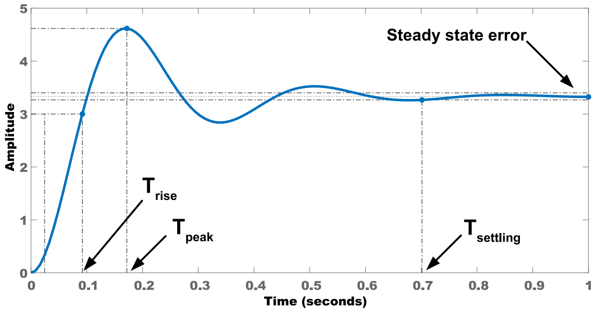


Figure 3.6: Step response of a sample control loop [BCP20].

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 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.

3.4.3 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

$$J = \int_0^\infty \left(x^T(t)Q_1x(t) + u^T(t)Q_2u(t)\right)dt, \tag{3.2}$$

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 [CPBM19] 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 [LC02], JitterTime [CPBM19] and TrueTime [HCÅ02] 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. (3.2) 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 [CPBM19].

3.5 Constraint Programming

The communication scheduling problem as a decision problem has been proved to be NP-complete in the strong sense [Sin07]. 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. 3.7, 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 the "CP model" box. Each variable has a finite set of values, called domain, that can be assigned to it (see Sect. 3.5.1). 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. 3.5.2 and evaluates them using the objective function defined in Sect. 3.5.3 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

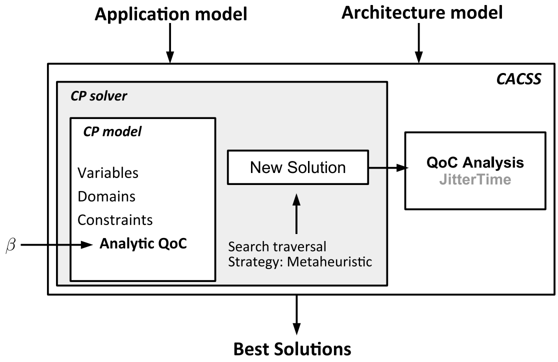


Figure 3.7: Overview of CACSS.

scale. Thus, we propose a novel *analytical model* for the QoC evaluation within the CP formulation, see Sect. 3.5.3. 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. 3.5.4.

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

$$\begin{aligned}
&\forall s_i \in \mathcal{S}, \forall m \in [1,..,|s_i|], \forall k \in [1,..,|r_j|],\\
&r_j = \mathbf{z}(s_i), \varepsilon_{v,w} = \mathbf{u}(r_j,k):\\
&f^k_{i,m}.l = \frac{s_i.c}{\varepsilon_{v,w}.s \times \varepsilon_{v,w}.mt}\\
&0 \leq f^k_{i,m}.\phi \leq (\frac{s_i.t}{\varepsilon_{v,w}.mt} - f^k_{i,m}.l)
\end{aligned} \tag{3.3}$$

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

3.5.2 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. (3.4).

$$\begin{aligned}
&\forall s_i, s_j \in \mathcal{S}, i \neq j, \forall m \in [1,..,|s_i|], \forall n \in [1,..,|s_j|],\\
&r_o = \mathbf{z}(s_i), \forall k \in [1,..,|r_o|], \quad r_p = \mathbf{z}(s_j), \forall l \in [1,..,|r_p|],\\
&\varepsilon_{v,w} = \mathbf{u}(r_o,k) = \mathbf{u}(r_p,l) :\\
&(f^k_{i,m}.\phi + m \times \frac{s_i.t}{\varepsilon_{v,w}.mt} \geq f^l_{j,n}.\phi + n \times \frac{s_j.t}{\varepsilon_{v,w}.mt} + f^l_{j,n}.l) \vee\\
&(f^l_{j,n}.\phi + n \times \frac{s_j.t}{\varepsilon_{v,w}.mt} \geq f^k_{i,m}.\phi + m \times \frac{s_i.t}{\varepsilon_{v,w}.mt} + f^k_{i,m}.l).
\end{aligned} \tag{3.4}$$

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. (3.5) 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 [IEE17].

$$\begin{aligned}
&\forall s_i \in \mathcal{S}, \forall m \in [1,..,|s_i|], \forall k \in [1,..,|r_j|),\\
&r_j = \mathbf{z}(s_i), \varepsilon_{v,w} = \mathbf{u}(r_j,k), \varepsilon_{w,x} = \mathbf{u}(r_j,(k+1)),\\
&\Delta = \varepsilon_{v,w}.d(s_i.c) + v_w.d(s_i.c) + \delta :\\
&f^{k+1}_{i,m}.\phi \times \varepsilon_{w,x}.mt \geq (f^k_{i,m}.\phi + f^k_{i,m}.l) \times \varepsilon_{v,w}.mt + \Delta.
\end{aligned} \tag{3.5}$$

We define the *Isolation constraint* in Eq. (3.6) 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 [COCS16] for more details. In Eq. (3.6), δ represents the network precision.

$$
\begin{aligned}
&\forall s_i, s_j \in \mathcal{S}, i \neq j, \forall m \in [1,..,|s_i|], \forall n \in [1,..,|s_j|],\\
&r_o = \mathbf{z}(s_i), \forall k \in (1,..,|r_o|], \quad r_p = \mathbf{z}(s_j), \forall l \in (1,..,|r_p|],\\
&\varepsilon_{v,w} = \mathbf{u}(r_o, k) = \mathbf{u}(r_p, l),\\
&\varepsilon_{a,v} = \mathbf{u}(r_o, k-1), \varepsilon_{b,v} = \mathbf{u}(r_p, l-1) :\\
&((f_{i,m}^{k}.\phi \times \varepsilon_{v,w}.mt + m \times s_i.t + \delta \leq\\
&f_{j,n}^{l-1}.\phi \times \varepsilon_{b,v}.mt + n \times s_j.t + \varepsilon_{b,v}.d(s_j.c)) \vee\\
&(f_{j,n}^{l}.\phi \times \varepsilon_{v,w}.mt + n \times s_j.t + \delta \leq\\
&f_{i,m}^{k-1}.\phi \times \varepsilon_{a,v}.mt + m \times s_i.t + \varepsilon_{a,v}.d(s_i.c))) \vee\\
&(s_i.p \neq s_j.p).
\end{aligned}
\tag{3.6}
$$

The *Deadline constraint* defined in Eq. (3.7) 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.

$$
\begin{aligned}
&\forall s_i \in \mathcal{S}, \forall m \in [1,..,|s_i|], r_j = \mathbf{z}(s_i),\\
&\varepsilon_{a,b} = \mathbf{u}(r_j, 1), \varepsilon_{y,z} = \mathbf{u}(r_j, |r_j|) :\\
&f_{i,m}^{1}.\phi \times \varepsilon_{a,b}.mt + s_i.d \geq \varepsilon_{y,z}.mt \times (f_{i,m}^{|r_j|}.\phi + f_{i,m}^{|r_j|}.l).
\end{aligned}
\tag{3.7}
$$

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. (3.8).

$$
\begin{aligned}
&\forall \gamma_i \in \Gamma, \forall s_j \in \gamma_i.\mathcal{I}, \forall s_k \in \gamma_i.\mathcal{O},\\
&\forall m \in [1,..,|s_j|], \forall n \in [1,..,|s_k|],\\
&r_o = \mathbf{z}(s_j), r_p = \mathbf{z}(s_k),\\
&\varepsilon_{a,b} = \mathbf{u}(r_o, |r_o|), \varepsilon_{b,z} = \mathbf{u}(r_p, 1),\\
&\Delta = \varepsilon_{a,b}.d(s_j.c) + \nu_b.d(s_j.c) + \delta :\\
&(f_{j,m}^{|r_o|}.\phi + f_{j,m}^{|r_o|}.l) \times \varepsilon_{a,b}.mt + \Delta \geq \varepsilon_{b,z}.mt \times f_{k,m}^{1}.\phi
\end{aligned}
\tag{3.8}
$$

3.5.3 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 a 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. 3.4 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 v_1. The flow s_1 is transmitted from the sensor node v_4 and routed via the switch v_3 to the node v_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 v_1 and routed via the switch v_2 to the actuator node v_5 and also has the same period as the control application.

The node v_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 v_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. (3.9), 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 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.

$$
\begin{aligned}
&\forall \gamma_i \in \Gamma, \forall s_j \in \gamma_i.\mathcal{I}, \forall s_k \in \gamma_i.\mathcal{O},\\
&\forall m,q \in [1,..,|s_j|], \forall n,u \in [1,..,|s_k|],\\
&r_o = \mathbf{z}(s_j), r_p = \mathbf{z}(s_k),\\
&\varepsilon_{a,b} = \mathbf{u}(r_o,|r_o|), \varepsilon_{e,f} = \mathbf{u}(r_o,1),\\
&\varepsilon_{c,g} = \mathbf{u}(r_p,|r_p|), \varepsilon_{b,z} = \mathbf{u}(r_p,1):\\
&\omega_1 = \sum \frac{f_{j,m}^{|r_o|}.\phi \times \varepsilon_{a,b}.mt}{s_j.t}\\
&\omega_2 = \sum \frac{s_k.t - f_{k,n}^{1}.\phi \times \varepsilon_{b,z}.mt}{s_k.t}\\
&\omega_3 = \sum \frac{|(f_{j,m}^{|r_o|}.\phi - f_{j,q}^{|r_o|}.\phi) \times \varepsilon_{a,b}.mt + (m-q) \times s_j.t|}{s_j.t}\\
&\omega_4 = \sum \frac{|(f_{k,n}^{1}.\phi - f_{k,u}^{1}.\phi) \times \varepsilon_{b,z}.mt + (n-u) \times s_k.t|}{s_k.t}\\
&\omega_5 = \sum \frac{|(f_{k,m}^{|r_p|}.\phi - f_{k,q}^{|r_p|}.\phi) \times \varepsilon_{c,g}.mt +}{}\\
&\frac{(f_{j,q}^{1}.\phi - f_{j,m}^{1}.\phi) \times \varepsilon_{e,f}.mt + (m-q) \times s_j.t|}{s_j.t}\\
&\Omega = \omega_1 + \omega_2 + \beta \times (\omega_3 + \omega_4 + \omega_5)
\end{aligned}
\tag{3.9}
$$

3.5.4 Search Strategy

In this work we used the Google OR-Tools [Goo21] 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 [BZP20] based on a Tabu Search metaheuristic algorithm [BK05], 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 $f_{i,m}^{k}.\phi$ 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 $f_{i,m}^{k}.l$.

3.6 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. 3.6.1 followed by comparing our proposed Control-Aware Communication Scheduling Strategy (CACSS) with the related work in Sect. 3.6.2. Afterwards, we evaluate our proposed method on the synthetic test cases in Sect. 3.6.3. In Sect. 3.6.4 we evaluate CACSS on a realistic test case. We also validate the generated GCLs using the *OMNET++* simulator in Sect. 3.6.5. Finally, we used the generated GCLs and validated them on a TSN hardware platform in Sect. 3.6.6.

3.6.1 Test Cases and Setup

We implemented CACSS in Java using Google OR-Tools [Goo21] 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. (3.10) where a and b are randomly chosen respectively from $[50, 100, 150]$ and $[100, 200, 300, 400]$. We have used Jitter-

Table 3.3: 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

bug for designing the control function K with the LQG control law [LC02] as discussed in Sect. 3.4.3. The test case sizes are progressively increasing in number of ESs, 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 3.3 where the sixth column shows the total number of flow frames.

$$G = \frac{a}{s^2 + b} \tag{3.10}$$

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. (3.10). The details of the realistic test case are shown in Table 3.6.

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 3.5. The hardware platform is presented in [SGJM$^+$20] and consists of three TSN switches that are connected in a daisy chain manner. The first and the last switches consist internal ESs. The links are full duplex with the speed of 1 Gbps and flows can be sent from both ESs. A schematic of the hardware platform is shown in Fig. 3.8 where the points for the measurement are marked.

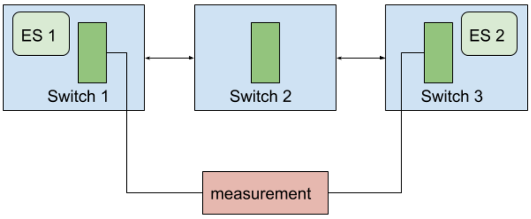


Figure 3.8: Schematics of the hardware platform.

3.6.2 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 [COCS16] and (ii) Frame-to-Window GCL (*FWGCL*) proposed in [SCS18a]. Table 3.4 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 [COCS16] 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 sched-

Table 3.4: 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

Table 3.5: 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

ules 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 [SCS18a], and since *FWGCL* introduces 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 3.3. 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.

3.6.3 Evaluation on Synthetic Test Cases

We evaluated the performance of CACSS on the synthetic test cases from Table 3.3. Our solution has successfully scheduled all the test cases and the schedules have zero

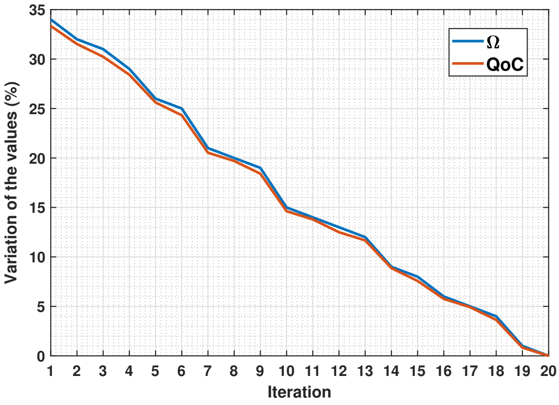


Figure 3.9: Comparison of analytical function Ω with QoC obtained by JitterTime for test case 5 from Table 3.3.

jitter. We first evaluate the runtime of our proposed solution. The solution runtime in milliseconds for each test case is given in column 10 in Table 3.3. 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. (3.9). 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 3.3 show the objective function value Ω Eq. (3.9) and QoC measured with JitterTime (which corresponds to the J value captured by Eq. (3.2)). 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 3.3, 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. (3.9) vs. the QoC value J obtained with a call to JitterTime. The results are shown in Fig. 3.9, 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.

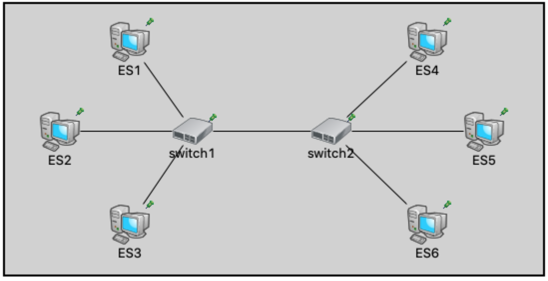


Figure 3.10: Implementation of TC1 in OMNET++.

3.6.4 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 3.6, 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).

3.6.5 OMNET++ validation

We have used the *OMNET++* simulator with the TSN NeSTiNg extension [FHC+19] 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. We took the synthetic test case 1 from Table 3.3, synthesized the GCLs with both CACSS and CACSS-ZFD. We simulated the schedules of all the synthetic and realistic test cases from Tables 3.3

Table 3.6: 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

and 3.6 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. 3.10 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 3.7, 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.

3.6.6 Evaluation on a Hardware Platform

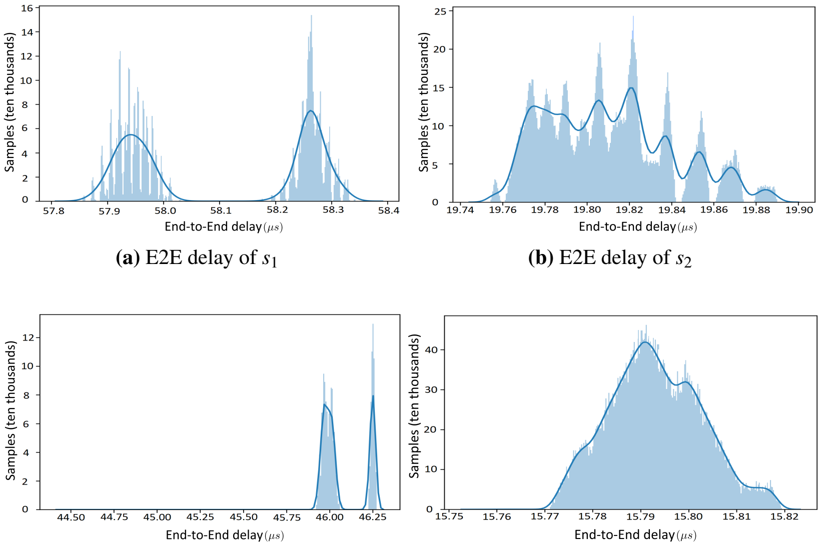
We have also evaluated the performance of CACSS on the hardware platform from [SGJM$^+$20] and in this context we removed the assumption that the forwarding delay is ignored. For this evaluation, we assumed that all the SWs are the same type as presented in [SGJM$^+$20]. The authors proposed the following equation for capturing the forwarding delay d in μs:

$$d = \lceil 4 + \frac{21 \times c}{400} \rceil, \tag{3.11}$$

where c is the size of the flows in bytes. Although we are using this TSN switch

Table 3.7: Simulation results of the synthetic test case 1 from Table 3.3

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



(a) E2E delay of s_1

(b) E2E delay of s_2

(c) E2E delay of s_3

(d) E2E delay of s_4

Figure 3.11: The details of the measured E2E delay of flows in the test case 2 form the Table 3.5 implemented on the hardware platform. Thick lines are kernel density estimates.

hardware implementation in a different application scenario compared to [SGJM$^+$20], 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. 3.8. We have used the three small “hardware test cases” from Table 3.5, 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 3.5 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 the deadlines or the control performance.

Let us illustrate the small variations measured in E2E delay for the hardware test case 2 from Table 3.5. Fig. 3.11 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. 3.5.2 in order to guarantee that deadlines are satisfied.

3.7 Related Work

There is already a lot of research on Fog Computing, focused mostly on aspects related to quality-of-service (QoS) [YLH$^+$18, MKB18, MNY$^+$18], with limited attention to safety-critical and real-time applications such as those used in the industrial areas. Real-time and safety-critical systems require guarantees for non-functional properties such as timing, e.g., that the deadlines are satisfied. Also, control applications have to fulfill non-functional properties related to control performance, e.g., QoC. Addressing the QoC for control applications in the Fog is still an open issue, researchers investigating the issue of degradation of control applications [ZHY16, ZSWY17, JX07]. For example, [MAS$^+$18] focuses on the routing and scheduling of messages of control applications to protect them form 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 [HZ19, SDD12, MYV$^+$04, PYKL11, BPZ02, SSS17] 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 [WS12, HZ19, Son09], 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 [MAS$^+$18] has considered routing and scheduling in Deterministic Ethernet, but lacks TSN-specific features which makes it difficult to implement the results, and uses an SMT formulation that cannot optimize the solutions and does not scale for large problem sizes. Our initial investigation in [MAS$^+$18, BZP20] 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 [DN16], 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 [WSJD15] increase the flexibility of the scheduling by employing a greedy-based 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 [AHM18], where authors aim to generate joint network routing and communication scheduling that are fault-tolerant, within a reasonably short time. Arestova et al. in [AHG20] 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 [PTO19], where multi-cast traffic and application distribution are allowed, and bandwidth is optimized. The same problem is addressed in [PO18] where a genetic algorithm is employed and in [GZRP18], where multiple traffic types are considered.

The use of SMT solvers for the communication scheduling is first proposed in [Ste10]. 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 [COCS16] 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 [COSS17], which introduces windows in GCLs and maps the frames to them. Another SMT model based on "array theory encoding" is proposed in [SCS18a], 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 [SCS18b].

The SMT-based schedulers are extended for the benefit of other applications. For example, in [PRCS16], 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 [CO16], 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 [PSS19] 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 [MAS^{+}19] where the authors aim to increase the resilience of the control applications to malicious interference.

3.8 Conclusions 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.

CHAPTER 4

Paper C: Extensibility-Aware Fog Computing Platform Configuration for Mixed-Criticality Applications

Fog Computing integrates applications with mixed-criticalities in a shared platform. Such applications need different approaches to guarantee their timing and dependability requirements. In this paper, we consider that critical control applications and Fog applications share a Fog Computing Platform (FCP). Critical control applications are implemented as periodic hard real-time tasks and messages and have stringent timing and safety requirements, and require safety certification. Fog applications are implemented as aperiodic tasks and messages and are not critical. We formulate an optimization problem for the joint configuration of critical control and Fog applications, such that (i) the deadlines and Quality-of-Control (QoC) of control applications are guaranteed at design-time, (ii) the configuration is extensible and supports the addition of additional future new control applications without requiring costly re-certification,

and (iii) the design-time configuration together with the runtime Fog resource management mechanisms, can successfully accommodate multiple dynamic responsive Fog applications. We evaluate our approach on several test cases assuming scenarios for hosting both Fog applications and future critical control applications. The results show that our approach generates extensible schedules which enables Fog nodes to handle Fog applications with a shorter response time and a larger number of future control applications.

4.1 Introduction

Industrial systems rely on Operational Technology (OT), which uses dedicated hardware and software in the computing platform, without support for dynamic changes and online re-configurations [Eur16]. OT uses different types of fieldbus communication solutions, e.g., Profibus [Comb] and EtherCAT [Coma], which are proprietary solutions that support real-time control and data integrity; and processing elements, e.g., Programmable Logic Controllers (PLCs) and Industrial Personal Computers (IPCs), which implement safety-related functions [JAAH18]. These technologies have limitations in the principles of operation and capacity [JAAH18] and are costly for future upgrades [Dec05]. Thus, the current OT infrastructure cannot deliver the innovations envisioned with Industry 4.0 [GVCL14]. Although Information Technology (IT), such as Cloud Computing and Artificial Intelligence (AI), can alleviate the limitations of OT, they cannot be used at the edge of the network, where industrial machines are located, and where very stringent nonfunctional properties have to be guaranteed [GVCL14]. Both industry and academia have made significant efforts towards the convergence of OT/IT, which is required to realize the vision of Industry 4.0, e.g., increase the connectivity, interoperability, and scalability of industrial systems [GLSC17]. This will bring increased productivity and flexibility, mass customization, reduced time-to-market, improved product quality, innovations, and new business models [BBM$^+$16]. Fog Computing is an architectural means to realize the OT/IT convergence [PRGS18].

Fog Computing has emerged as a promising paradigm for enabling applications in various domains such as connected vehicles [BMZA12] and the Industrial Internet of Things (IIoT) [CLL18]. According to this paradigm, the applications that demand guarantees in safety, security, and real-time performance, are executed on a Fog Computing Platform (FCP), which extends from the Cloud towards the edge of the network. Other terms (e.g., *Edge Computing*) with similar objectives and principles are also used for such platforms [Ope21a]. An FCP includes nodes capable of communicating and executing computations, i.e., Fog Nodes (FNs), in the proximity of the "things" (e.g., machines) and data sources [KSLP19] to guarantee effective collaboration between the devices, nodes, and the Cloud (see Fig. 4.1). An FN is a compute node that needs to integrate mixed-criticality applications that share the FCP, with different timing-

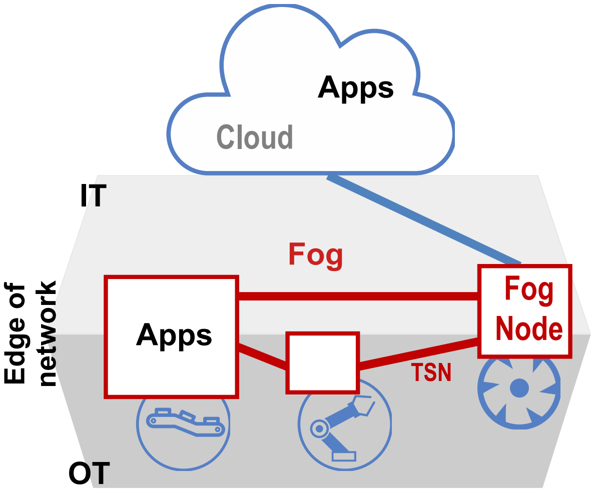


Figure 4.1: Fog Computing Platform: FNs (boxes), equipment and the Cloud are connected via network (thick lines). Mixed-criticality applications are running on FNs and in the Cloud [BP21a]

and safety-criticalities [BMZA12, BD18]. Hence, the challenge of handling mixed-criticality applications on FCPs has to be addressed [LYD$^+$17].

Mixed-criticality applications can be classified in several ways depending, e.g., on their safety-criticality and time-criticality [BD18]. In safety-critical systems a failure may lead to loss of life or damage to the environment or property. Such systems require pre-release certification that employs safety assurance processes to show that the risks have been properly identified and mitigated. Many safety-critical systems are also real-time, where the correctness of the results depends also on the time when they are delivered, e.g., deadlines have to be satisfied. Regarding time-criticality, real-time applications can be hard real-time, where missing a deadline is considered a failure or soft real-time, where deadline misses degrade the quality-of-service of the applications [Kop11].

At one extreme, we have the safety-critical real-time systems that control industrial process and have to be operational even in the case of failure. Due to their safety nature, such applications have to be certified. Safety Integrity Levels (SILs) are assigned to the different safety functions in a system and dictate the assurance and development processes that have to be used. Certification standards require that safety functions of different criticality levels are protected (isolated), so they cannot influence each other [Int10], and the resources needed for their operation have to be statically assigned pre-release. Any changes in their functionality or configuration will trigger a costly re-certification. We call these applications “critical control applications”, and they also need guaranteed control performance, captured via Quality of Control (QoC) metrics. They are implemented using periodic hard real-time tasks running on FNs and messages (transmitted as flows on the network) and they are allocated resources via a static design-time configuration on the FNs and the network. System engineers typically overprovision resources for these applications, which is costly, to allow the addition of new safety functions without modifying the existing configurations.

At the other extreme, we have non-critical dynamic applications that do not have stringent timing requirements. We make the key observation that to realize the vision of Industry 4.0, such non-critical dynamic applications, e.g., related to new business model, data analytics, software updates for security and maintenance, and connected equipment services, which have not been considered at design-time, have to be hosted by an FCP. Such “Fog applications” implement the innovative functionality needed to realize Industry 4.0 on the converged OT/IT Fog infrastructure, without jeopardizing the performance and safety of the critical control applications. These Fog applications are aperiodic, i.e., their arrival-time is unknown, but the FCP should dynamically allocate resources such that their quality-of-service (QoS) is maximized, e.g., their response times are reduced. As a result, there is a tension between the resource requirements of Fog applications that are only known at runtime, and those of critical control applications that have to be allocated at design-time.

This paper assumes that the OT/IT convergence has taken place via FCPs such as [PZB$^+$21, TTT21, Neb21], and hence FNs now replace the processing elements within the industrial domain. For example, in [Neb21] FNs are used in robotic use cases and can handle mixed-criticality applications. The FNs communicate with each other to implement the required functionality and share the workload of the mixed-criticality applications, and they communicate with the Cloud for exploiting its available resources [Kar19]. We also assume that the FCP uses IEEE 802.1 Time-Sensitive Networking (TSN) [IEE21b] as a deterministic communication solution for communication, as envisioned by several industrial consortia [PRGS18]. 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. TSN supports multiple traffic types, and hence, is suitable for mixed-criticality applications running on an FCP.

As we discuss in Sect. 4.6 that covers the related work, researchers have proposed a plethora of methods for the configuration of, on the one hand, critical control applications [BD18], and, on the other hand, Fog applications [PML$^+$19]. However, their joint configuration has received limited attention [BCP20]. The critical control applications have to be assigned resources at design-time not only such that they meet their stringent real-time and QoC requirements, but also that at runtime the Fog applications can be easily accommodated.

Researchers have also considered the issue of extensibility in the context of Cyber-Physical System (CPS) [WCP$^+$05, PEPP04], i.e., statically configuring a CPS platform such that resources are available periodically, e.g., slack in schedule tables, to accommodate future applications without changing the existing configuration. Keeping existing configurations (e.g., assignment of tasks, routes of flows and their schedules) unmodified is desirable since it preserves the performance level of the critical control applications [BCP20] and does not require a costly re-certification of safety-critical applications. However, the extensibility has not been addressed yet for FCPs that also need to accommodate Fog applications at runtime.

In this paper we address the configuration of an FCP to support both critical control applications and Fog applications. We consider these two extremes because they will drive solutions that can support a wider range of applications, e.g., real-time applications that are not safety-critical, or Fog applications that may be periodic soft real-time and benefit from maximizing their QoS. As mentioned, applications are composed of tasks that interact via messages. We consider that the critical control applications are configured at design-time and use static cyclic scheduling for both tasks and messages. Such applications use redundancy to tolerate failures. Adding redundancy to a system is orthogonal to the problems considered in the paper, e.g., by replicating hardware or using rollback recovery for tasks [PIEP09] and by using re-transmission or redundant routes for messages [RPC20]. We decide the mapping of tasks to the cores of the multicore FNs, the routing of flows, and the schedule tables for both tasks and messages. The Fog applications are configured at runtime using migration mechanisms which will decide the mapping of tasks and flows to the resources of the FCP available after the configuration of control applications.

Contributions: The contributions of this paper are as follows. We motivate the need for a novel configuration optimization approach for mixed-criticality applications running on an FCP. We assume that the platform uses partitioning to enforce the spatial and temporal isolation between applications with different criticalities [BCP20]. We use a hierarchical scheduling model (Sect. 4.2.4) that can accommodate multiple scheduling policies, targeting the different time-criticality requirements of applications. The critical control applications are scheduled using static cyclic scheduling (i.e., they are time-triggered) and the resources of the Fog applications are allocated at runtime via fixed-priority servers that are dimensioned at design-time jointly with the critical application configurations. We consider that the critical control applications use Scheduled Traffic (ST) for their flows, implemented via IEEE 802.1Qbv, which defines a Time-Aware Shaper (TAS) mechanism that enables the scheduling of flows based on a global schedule table. The flows of the Fog applications use Strict Priority (SP) flows that are sent with lower priority in the gaps of the ST traffic schedule tables. We formulate an optimization problem for the joint configuration of critical control and Fog applications, such that (1) the deadlines and QoC of control applications are guaranteed at design-time, (2) the design-time FCP configuration is extensible and supports the addition of a larger number of future new control applications, and (3) the design-time configuration together with the runtime Fog resource management mechanisms, can successfully accommodate responsive Fog applications.

We propose a Constraint Programming (CP)-based optimization strategy to synthesize such optimized configurations. Synthesizing a design-time configuration means deciding: (i) the mapping of critical control tasks to the cores of the FNs, (ii) the routing of critical control flows, (iii) the schedule tables for critical control tasks and flows, (iv) the slack in these schedule tables to increase their flexibility, and (v) the period and budget of the fixed-priority servers that allocate resources at runtime to the Fog applications. At runtime, our approach handles (vi) the migration of Fog tasks to the FNs that have resources for their execution, (vii) the scheduling of Fog tasks on the servers and of flows on TSN. We evaluate our CP approach on several synthetic and realistic test cases.

The rest of the paper is structured as follows. Sect. 4.2 presents our underlying system model, and Sect. 4.3 presents the problem formulation. Afterwards, we present our configuration approach in Sect. 4.4, and evaluate the performance of our proposed approach in Sect. 4.5. Finally, Sect. 4.6 describes the related work, and Sect. 4.7 concludes the paper.

4.2 System Models

The FCP model is presented in Sect. 4.2.1, and in Sect. 4.2.2 we present how TSN works for scheduled and strict priority traffic. The critical control and Fog application models are presented in Sect. 4.2.3 and Sect. 4.2.4 presents the scheduling policies used to jointly schedule these mixed-criticality applications. Table 4.1 summarizes the notation.

4.2.1 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 can be an End-System (ES) (e.g., an FN, a sensor, an actuator, or a machine), 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. All nodes have input (ingress) and output (egress) ports. We denote the set of egress ports of a node with $v_i.P$. A port $p_j \in v_i.P$ is linked to at most one other node. We denote the set of all FNs with $\mathcal{N} \subset \mathcal{V}$. Each FN $N_i \in \mathcal{N}$ has a multicore processor and each core is denoted by $\mathcal{C}_j \in N_i.C$.

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 $\varepsilon_{i,j} \in \mathcal{E}$ and $\varepsilon_{j,i} \in \mathcal{E}$; a

Table 4.1: Summary of the notation

Symbol	Notation
$\mathcal{G}$	Architecture graph
$v_i \in \mathcal{V}$	Architecture node
$N_i \in \mathcal{N}$	Fog Node
$\mathcal{C}_j \in N_i.C$	A core
$p_j \in v_i.P$	Egress port
$q_j \in p_i.Q$	Priority queue
$\varepsilon_{i,j} \in \mathcal{E}$	Link
$\varepsilon_{i,j}.s$	Link speed
$\varepsilon_{i,j}.d(c)$	Link propagation delay
$r_i \in \mathcal{R}$	Route
$\lvert r_i \rvert$	Number of links in a route
H	Hyperperiod
W_{p_i}	Periodic window
$W_{p_i}.c$	Window capacity
$W_{p_i}.t$	Window period
$W_{p_i}^{j}.\phi$	Window slice offset
$\gamma_i \in \Gamma$	Critical control application
$\tau_j \in \gamma_i.\mathcal{T}$	Critical control task
$\tau_j.t$	Critical control task period
$\tau_j.d$	Critical control task deadline
$\tau_j.c$	Critical control task WCET
$\tau_j^k.\phi$	Critical control job offset
$f_j \in \gamma_i.\mathcal{F}$	Critical control flow
$f_j.p$	Critical control flow priority
$f_j.c$	Critical control flow size
$f_j.t$	Critical control flow period
$f_j.d$	Critical control flow deadline
$\lvert f_i \rvert$	Number of critical control frames
$f_{i,m}^{k}.\phi$	Critical control frame offset
$\mathcal{M}$	The task mapping function to the fog nodes
$\gamma_i' \in \Gamma'$	Fog application
$\tau_j' \in \gamma_i'.\mathcal{T}$	Fog task
$\tau_j'.c$	Fog task workload
$\tau_j'.j$	Fog task arrival time
$f_j' \in \gamma_i'.\mathcal{F}$	Fog flow
$f_j'.c$	Fog flow size
$f_j'.j$	Fog flow arrival time
$D_{\mathcal{C}_i}$	Defferable server
$D_{\mathcal{C}_i}.c$	Server capacity
$D_{\mathcal{C}_i}.t$	Server period
$D_{\mathcal{C}_i}^{j}.\phi$	Server slice offset

link is attached to one port of the node v_i and one port of the node v_j.

Each link $\varepsilon_{i,j}$ is characterized by the tuple $\langle s, d \rangle$ denoting the speed of the link in Mbit/s and the propagation delay function of the link in ms. The propagation delay function of a frame on a link $\varepsilon_{i,j}.d(size)$ is calculated based the frame's size, the physical medium and the link length. The function d is a notation used in the constraints in Sect. 4.4.2 and it is attached to the link concept, i.e., $\varepsilon.d(size)$ means $d(\varepsilon, size)$.

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 node, and ending with a link to a listener node. 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 $\mathbf{u} : \mathcal{R} \times \mathbb{N}_0 \to \mathcal{E}$ to capture the j_{th}

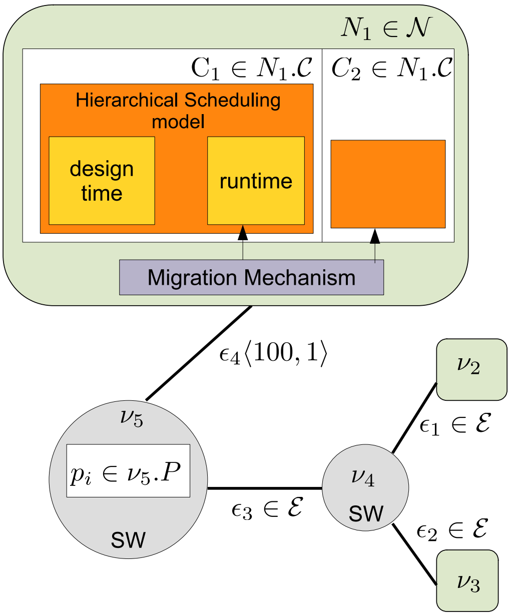

$N_1 \in \mathcal{N}$
$C_1 \in N_1.\mathcal{C}$
$C_2 \in N_1.\mathcal{C}$
Hierarchical Scheduling model
design time
runtime
Migration Mechanism
$\epsilon_4\langle 100, 1\rangle$
ν_2
ν_5
$p_i \in \nu_5.P$
SW
$\epsilon_1 \in \mathcal{E}$
ν_4
SW
$\epsilon_3 \in \mathcal{E}$
$\epsilon_2 \in \mathcal{E}$
ν_3

Figure 4.2: Example architecture with three end-systems and two switches: The orange box shows the hierarchical scheduling model; the purple box shows the migration mechanism [ATD19].

link of the route r_i.

An architecture model with three ESs (consisting of one FN and two sensors) and two SWs is presented in Fig. 4.2, where the thick lines are physical links. We also show in the figure examples on how the notation is used. The FCP is envisioned to host mixed-criticality applications which have to be separated from each other [PZB+21]. The separation is realized via temporal and spatial partitioning implemented via hypervisors [BCP20] which has not been considered in this paper. However, separation can be introduced as additional constraints in our proposed CP model.

Additionally, the FCP is envisioned to handle Fog applications via monitoring and resource management techniques [PZB+21]. Once Fog applications are submitted to run on the FCP, the *Fog controller* FN, which is determined at runtime using mechanisms such as [KP17], receives the submission request. The Fog controller has knowledge on the available resources on the FNs and SWs using the resource discovery algorithms such as [KP17, Ope21b, Eur21] at runtime. It can decide on the placement of application tasks on the FNs using a decentralized resource allocation technique [ATD19, SKR+18] which determines the FN that provides the minimum response time for the application using response time analysis such as [Alm03]. The fog

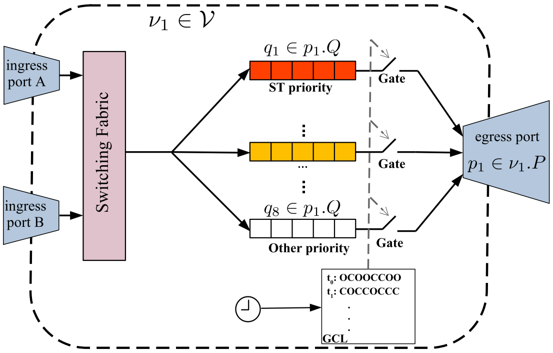


Figure 4.3: TSN switch internals [BP21a].

controller also decides the routing of flows using a routing algorithm such as [GZPS17] considering the available resources on SWs at runtime. The scheduling policies used in the FCP are discussed in Sect. 4.2.4.

4.2.2 TSN Switch Model

A TSN switch consists of ingress ports, a switching fabric, priority queues, gates, a Gate Control List (GCL) and egress ports, see Fig. 4.3. The switching fabric receives flows from the ingress ports and forwards each flow to the egress port p_i which is determined by the predefined internal routing table. The egress port which has a set of eight priority queues $p_i.Q$ (according to the IEEE 802.1Q standard [IEE14]), 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 is used for the ST and the remaining queues are used for the less critical traffic, similar to [COCS16]. 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. The scheduling relies on a clock synchronization mechanism 802.1ASrev [IEE17], which defines a global notion of time.

Flows of critical applications are transmitted using ST based on GCLs and have a higher priority. 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 [COCS16] for details. Flows of Fog applications are lower priority and are sent based on their priorities in the intervals of time (also called windows) when the respective queue gates are open, i.e., when ST flows are not scheduled.

To this end, we define a periodic window W_{p_j} on each port of the network nodes $p_j \in v_i.P$. Each window is characterized by the tuple $\langle c,t \rangle$ denoting the capacity and period of the window in ms. Additionally, each window will have several instances which are referred to as window slices, in a hyperperiod H (which is the system cycle, see Sect. 4.2.3). We associate each window slice $W_{p_i}^{j}$ with its start time ϕ.

4.2.3 Application Model

There have been several application models proposed in the literature, depending on the periodicity and time-criticality of the applications [Kop11]. Our application model consists of (i) a set of critical control applications considered at design-time, denoted with Γ, which we capture using a periodic hard real-time task model, and (ii) a set of Fog applications considered at runtime, denoted with Γ', for which we use an aperiodic best-effort task model.

Critical control applications consist of tasks which exchange massages and implement control functions for controlling dynamical systems, see [BCP20] for more details. All tasks and messages in a critical control application are periodic and have the same period. Thus, we define a hyperperiod H which is a system cycle and equal to the least common multiple of all application periods. Each critical control application $\gamma_i \in \Gamma$ is modeled with a Directed Acyclic Graph (DAG), where nodes represent tasks and edges represent data flows between the nodes. The set of all tasks and the set of all flows in a critical control application are denoted with $\gamma_i.\mathcal{T}$ and $\gamma_i.\mathcal{F}$ respectively.

A critical task $\tau_j \in \gamma_i.\mathcal{T}$ is characterized by the tuple $\langle t,d,c \rangle$ denoting the task period, the task deadline and a known Worst-Case Execution Time (WCET) on the mapped FN in ms. Each task is ready to execute when all its inputs have arrived. The output of a task is produced upon the termination of the task. The mapping of tasks to the cores of FNs is captured by the function $\mathcal{M}$ which is determined by our proposed scheduling algorithm. The task τ_i will have $H/\tau_i.t$ instances denoted with $|\tau_i|$ in a hyperperiod H which are referred to as jobs denoted with τ_i^j. A job is associated with ϕ denoting the start time of the job.

A critical flow $f_i \in \gamma_i.\mathcal{F}$ is responsible for sending the frames that encapsulate the data from an application and it is characterized by the tuple $\langle p,c,t,d \rangle$ denoting the priority, the size in bytes, the period in ms, inherited from the originating task period, and the flow deadline, i.e., the maximum allowed end-to-end delay in ms. Depending on the 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

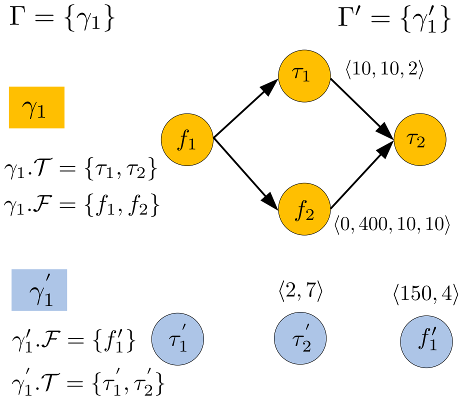

$\Gamma = \{\gamma_1\}$
$\Gamma' = \{\gamma'_1\}$
γ_1
$\gamma_1.\mathcal{T} = \{\tau_1, \tau_2\}$
$\gamma_1.\mathcal{F} = \{f_1, f_2\}$
f_1
τ_1
$\langle 10, 10, 2\rangle$
τ_2
f_2
$\langle 0, 400, 10, 10\rangle$
γ'_1
$\gamma'_1.\mathcal{F} = \{f'_1\}$
$\gamma'_1.\mathcal{T} = \{\tau'_1, \tau'_2\}$
τ'_1
$\langle 2, 7\rangle$
τ'_2
$\langle 150, 4\rangle$
f'_1

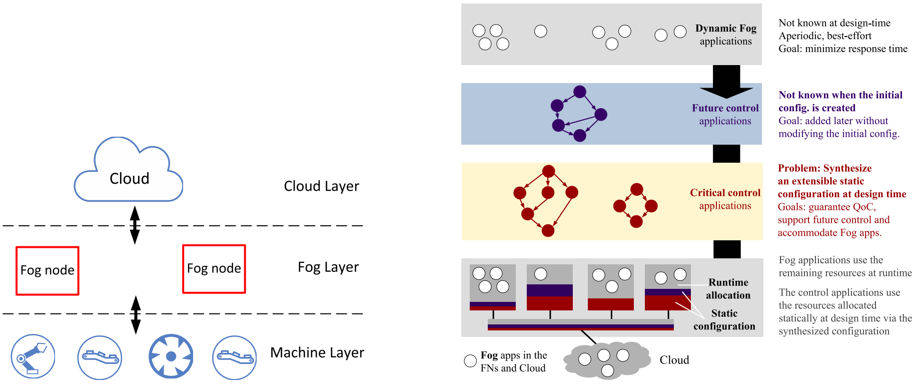
Figure 4.4: Example application model with one critical control application and one Fog application.

number of instances for a flow f_i is denoted with $|f_i|$, and is derived from the period of the flow t and the hyperperiod H.

Each flow f_i is transmitted via a route r_j which is captured by the function $\mathbf{z} : \mathcal{F} \to \mathcal{R}$ 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 define a frame for each instance $1 \leq m \leq |f_i|$ of the flow f_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 ϕ denoting the start time of the frame.

Each Fog application $\gamma'_i \in \Gamma\prime$ consists of a set of aperiodic tasks and a set of aperiodic flows denoted by $\gamma'_i.\mathcal{T}$ and $\gamma'_i.\mathcal{F}$, respectively. The tasks do not have data dependencies, i.e., a task will start when it arrives, but they may exchange data asynchronously using flows. A Fog task $\tau'_j \in \gamma'_i.\mathcal{T}$ is denoted by the tuple $\langle c, j\rangle$ denoting the workload (which is the average execution time) on the mapped core of FNs in ms and its arrival time in ms. The arrival times and workloads of Fog tasks are unknown at design time. A Fog flow $f'_j \in \gamma'_i.\mathcal{F}$ is also aperiodic. Such a flow is denoted by the tuple $\langle c, j\rangle$ denoting the size in bytes and the arrival time in ms.

An example application model composed of two applications is shown in Fig. 4.4. γ_1 is a critical control application and γ'_1 is a Fog application. The task and flow details are given in the figure.


Cloud
Cloud Layer
Fog node
Fog node
Fog Layer
Machine Layer
Dynamic Fog applications
Not known at design-time
Aperiodic, best-effort
Goal: minimize response time
Future control applications
Not known when the initial config. is created
Goal: added later without modifying the initial config.
Critical control applications
Problem: Synthesize an extensible static configuration at design time
Goals: guarantee QoC, support future control and accommodate Fog apps.
Runtime allocation
Static configuration
Fog applications use the remaining resources at runtime
The control applications use the resources allocated statically at design time via the synthesized configuration
Fog apps in the FNs and Cloud
Cloud

(a) Example use case with an FCP hosting mixed-criticality applications.

(b) Optimization problem overview

Figure 4.5: An FCP hosting Fog applications and future control applications at run-time.

4.2.4 Scheduling Policies

Mixed-criticality applications require different scheduling policies depending on their timing criticality [But11]. Similar to related work, we use static cyclic scheduling (timeline scheduling) for critical control applications, since this is a scheduling policy suited for hard real-time applications in safety-critical areas. Fog applications are scheduled using a deferrable server [SLS95]. To put together several scheduling policies we use the hierarchical scheduling model [SL08], which consists of several levels of schedulers, starting at level 1 where a single scheduler reserves resources to applications and schedulers at the next level, i.e., level 2 [SL03]. Because the hierarchical scheduling model is general, we can accommodate any combination of schedulers needed by the mixed-criticality applications, in-between the two extremes: the static scheduling of critical control applications versus the dynamic scheduling of Fog applications.

Our scheduling model employs a static scheduler at level 1 for scheduling control applications and a periodic fixed priority server at level 2. The scheduler at level 1 uses a static cyclic scheduling, also known as time-triggered or timeline scheduling [But11]. A static cyclic schedule captures the start and finishing time of tasks and flows and repeats with the hyperperiod H.

A fixed priority server is implemented as a periodic task $D_{\mathcal{C}_i}$ that runs in a core $\mathcal{C}_i$ and it is characterized by the tuple $\langle c,t \rangle$ denoting the capacity of the server and the period of the server in ms. Within a hyperperiod H, a server will have several instances which are referred to as server slices. Since level 1 uses static cyclic scheduling, the servers have to be scheduled in the static schedule at design-time together with the control tasks. Each server slice is denoted by $D^j_{C_i}$ and is associated with ϕ denoting its start time.

We consider that the aperiodic Fog tasks are served by such servers, and we use a deferrable server [SLS95], which uses soft resource reservation techniques to allocate its resources to the Fog tasks. Fog application flows are transmitted using the TSN mechanisms specific for SP flows in the windows when ST critical flows are not scheduled, see Sect. 4.2.2.

The servers and the future applications will use the slack introduced in the schedule tables. We refer to the slack with the same notation used for the fixed priority server, and use them interchangeably.

4.3 Problem Definition

We formally define the extensibility-aware configuration problem we address in the paper as follows, see Fig. 4.5b for an illustration. Given (1) a set of critical control applications Γ, (2) an FCP modeled with an architecture graph $\mathcal{G}$, we want to determine a configuration Ψ consisting of: (i) the mapping $\mathcal{M}$ of critical control tasks to the cores of the FNs, (ii) the set of routes $\mathcal{R}$ of critical control flows, (iii) the static task schedule tables, (iv) the GCLs, (v) the period and capacity of port windows W_{p_i}, and (vi) the period and capacity of the deferrable servers D_{C_i}. At runtime, our approach handles (vii) the migration of Fog tasks to the FNs that have resources for their execution, (viii) the scheduling of Fog tasks on the servers and of their flows on TSN.

We are interested in an optimized configuration Ψ such that: the deadlines of all the critical control applications are met, the QoC of control applications, as defined in Sect. 4.4.3, is maximized, and the extensibility of the configuration Ψ is maximized, which supports adding future control applications without modifications to Ψ and enables shorter response times for Fog applications.

Given a mapping $\mathcal{M}$, synthesizing a static task schedule for the cores of FNs is equivalent to determining (i) the critical control task offsets $\tau^j_i.\phi$, and (ii) the server slices' offsets $D^j_{C_i}.\phi$. Additionally, synthesizing GCLs for the ports of network nodes is equivalent to determining (i) the critical frames' offsets $f^k_{i,m}.\phi$, and (ii) the window offsets $W^j_{p_i}.\phi$.

As a motivational example, let us consider an industrial use case with several robots that uses an FCP architecture (see Fig. 4.5a). The critical control applications are responsible for the operation of robots, whose tasks and flows are implemented on the FNs and TSN, respectively; the configuration for these tasks and flows is determined at design-time. This static configuration is designed to ensure the correct functionality of the safety-critical virtualized control on the FCP. In contrast to OT which uses over-provisioned dedicated hardware and software to run implement the control, an FCP platform should also be able to host Fog applications.

We consider that the system engineers would like to deploy a set of innovative Fog applications for data analytics to optimize the industrial processes. They are deployed on the FCP via one of the FNs or from the Cloud. Also, we assume that at some time in the future, there is a need for extending the robot control, e.g., via an updated pathfinder algorithm, see Fig. 4.5b. To minimize safety assurance costs, the new robot control should be added to the FCP without modifying the existing configuration, since modifications may trigger a costly re-certification. In addition, the Fog applications must run on the FCP at runtime without modifying the statically defined configuration for the control tasks and flows. Therefore, the static configuration should be extensible: resources for possible extensions of critical control applications have to be added pre-release and the resources that Fog applications use at runtime should be allocated at design-time.

4.4 Proposed Solution

In this section, first, we describe our configuration optimization approach in detail and present an overview of the objective, a valid solution, and the extensibility metrics. Afterwards, we present the CP model and the CP constraints we use to solve the problem, in Sect.s 4.4.1 4.4.2, respectively. Finally, we define the objective function in Sect. 4.4.3 and present our heuristic search methodology in Sect. 4.4.4.

Just the scheduling problem, in even simpler contexts, has been proved to be Non-deterministic Polynomial time (NP)–complete in the strong sense [Sin07, Ull75]. We propose an optimization strategy called Extensible Configuration Optimization Strategy (ECOS), based on a CP formulation that uses search heuristics inside the CP solver, aiming at finding solutions even for large problem sizes. Fig. 4.6 presents an overview of ECOS which takes as the inputs the architecture model, and the application model; and outputs a set of the best solutions found during search.

As mentioned, ECOS is based on a CP formulation. 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. Each variable has a finite set of values, called domain, that can be assigned to it (see Sect. 4.4.1). Constraints restrict the variables'

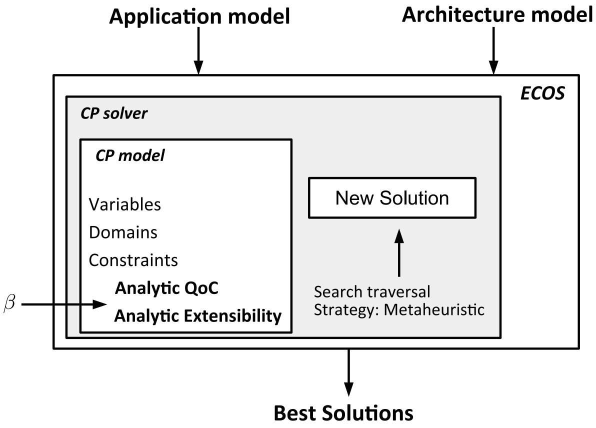


Figure 4.6: Overview of ECOS.

domains by bounding them to a range of values and defining relations between the domains of different variables (see Sect. 4.4.2). The search aims to find good quality solutions in a reasonable time, but it does not guarantee the optimality.

ECOS visits solutions that satisfy the constraints defined in Sect. 4.4.2 which are all valid solutions; and evaluates them using the objective function defined in Sect. 4.4.3 to check if the solution is an *improving* solution, i.e., better than the best solutions found so far. The objective function introduces metrics for extensibility and QoC. In an extensible solution, the slack is distributed in a way that is periodic, uniform and starts early within a period. Moreover, in a good QoC solution, the input-output jitter of a critical control application, i.e., variation in the end-to-end response of the application, is minimized.

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. 4.4.4.

4.4.1 CP model

We define seven sets of decision variables for the CP model. Each decision variable is associated with a domain from which the CP solver decides the variable's value. The decision variables and their domain for flows and windows are defined by

$$
\begin{aligned}
&\forall \gamma_i \in \Gamma, \forall f_j \in \gamma_i.\mathcal{F}, \forall m \in [1,..,|f_j|],\\
&\forall l \in [1,..,|r_k|], r_k = \mathbf{z}(f_j), \varepsilon_{v,w} = \mathbf{u}(r_k, l):\\
&0 \leq f^k_{j,m}.\phi \times \varepsilon_{v,w}.s \leq (f_j.t \times \varepsilon_{v,w}.s - f_j.c)\\
&\forall v_i \in \mathcal{V}, \forall p_j \in v_i.P, \forall m \in [1,..,H/W_{p_j}.t]:\\
&0 \leq W^m_{p_j}.\phi \leq (W_{p_j}.t - W_{p_j}.c)
\end{aligned}
\tag{4.1}
$$

where, the frame offsets' domain is the range of 0 to the time point when the frame has enough time to be transmitted within the frame period. The domain for window slice offsets is in the range of 0 to the time point when the slice accesses its capacity within its period.

For task scheduling, the decision variables are associated with the job offsets and the server slice offsets on cores of FNs. The variables and their domains are defined in Eq. (4.2) where the job offsets' domain is in the range from 0 to the time point when a duration equal to job's WCET is left within its period. Similarly, the domain for server slice offsets is in the range of 0 to the time point when the slice accesses its capacity within its period.

$$
\begin{aligned}
&\forall \gamma_i \in \Gamma, \forall \tau_j \in \gamma_i.\mathcal{T}, \forall m \in [1,..,|\tau_i.t|]:\\
&0 \leq \tau^m_j.\phi \leq (\tau_j.t - \tau_j.c)\\
&\forall N_i \in \mathcal{N}, \forall \mathcal{C}_j \in N_i.C, \forall m \in [1,..,H/D_{\mathcal{C}_j}.t]:\\
&0 \leq D^m_{\mathcal{C}_j}.\phi \leq (D_{\mathcal{C}_j}.t - D_{\mathcal{C}_j}.c)
\end{aligned}
\tag{4.2}
$$

The mapping function $\mathcal{M}$ which captures the mapping of tasks to the cores of FNs is also defined as a decision variable. The domain and the co-domain of the function is defined in Eq. (4.3) where the function domain is the set of all tasks in the application model and the function co-domain is the set of all cores of FNs.

$$
\begin{aligned}
&\mathcal{M} : X \longrightarrow Y :\\
&X = \{\tau_i | \tau_i \in \gamma_j.\mathcal{T}, \gamma_j \in \Gamma\} \quad Y = \{\mathcal{C}_i | \mathcal{C}_i \in N_j.C, N_j \in \mathcal{N}\}
\end{aligned}
\tag{4.3}
$$

4.4.2 CP Constraints

Flow Constraints: We define five constraints similar to [BP21a] that regulate the network traffic and relate the domain of the CP variables. CP only finds the feasible solutions, i.e., those where all the constraints are met.

The *Link Overlap constraint* imposes the restriction on the solution which avoids sharing a physical link resources with more than one frame or window slice at any time. The constraint is defined in Eq. (4.4). Note that as presented in Sect. 4.2.1, each link is attached to one port of the source and one port of the destination. Thus, a port is equivalent to a link.

$$\begin{aligned}
&\forall \gamma_i, \gamma_j \in \Gamma, \forall f_k \in \gamma_i.\mathcal{F}, \forall f_l \in \gamma_j.\mathcal{F}, k \neq l,\\
&\forall m \in [1,..,|f_k|], \forall n \in [1,..,|f_l|],\\
&r_o = \mathbf{z}(f_k), \forall q \in [1,..,|r_o|], \quad r_p = \mathbf{z}(f_l), \forall x \in [1,..,|r_p|],\\
&p_{v,w} \equiv \varepsilon_{v,w} = \mathbf{u}(r_o,q) = \mathbf{u}(r_p,x),\\
&\forall y \in [1,...,H/W_{p_{v,w}}.t]:\\
&((f^{q}_{k,m}.\phi + m \times f_k.t \geq f^{x}_{l,n}.\phi + n \times f_l.t + \frac{f_l.c}{\varepsilon_{v,w}.s}) \vee\\
&(f^{x}_{l,n}.\phi + n \times f_l.t \geq f^{q}_{k,m}.\phi + m \times f_k.t + \frac{f_k.c}{\varepsilon_{v,w}.s})) \wedge\\
&((f^{q}_{k,m}.\phi + m \times f_k.t \geq W^{y}_{p_{v,w}}.\phi + y \times W_{p_{v,w}}.t + W_{p_{v,w}}.c) \vee\\
&(W^{y}_{p_{v,w}}.\phi + y \times W_{p_{v,w}}.t \geq f^{q}_{k,m}.\phi + m \times f_k.t + \frac{f_k.c}{\varepsilon_{v,w}.s}))
\end{aligned} \tag{4.4}$$

The *Route constraint* enforces the ordered propagation of a frame concerning its associated route from its talker all the way to its listener. We define the constraint in Eq. (4.5) 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 [IEE17].

$$\begin{aligned}
&\forall \gamma_i \in \Gamma, \forall f_j \in \gamma_i.\mathcal{F}, \forall m \in [1,..,|f_j|], \forall k \in [1,..,|r_l|),\\
&r_l = \mathbf{z}(f_j), \varepsilon_{v,w} = \mathbf{u}(r_l,k), \varepsilon_{w,x} = \mathbf{u}(r_l,(k+1)):\\
&f^{k+1}_{j,m}.\phi \geq f^{k}_{j,m}.\phi + \frac{f_j.c}{\varepsilon_{v,w}.s} + \varepsilon_{v,w}.d(f_j.c) + \delta.
\end{aligned} \tag{4.5}$$

We define the *Isolation constraint* in Eq. (4.6) 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 other words, 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 [COCS16] for more details. In Eq. (4.6), δ represents the network precision.

$$\begin{aligned}
&\forall \gamma_i, \gamma_j \in \Gamma, \forall f_k \in \gamma_i.\mathcal{F}, \forall f_l \in \gamma_j.\mathcal{F}, k \neq l,\\
&\forall m \in [1,..,|f_k|], \forall n \in [1,..,|f_l|],\\
&r_o = \mathbf{z}(f_k), \forall q \in (1,..,|r_o|], \quad r_p = \mathbf{z}(f_l), \forall x \in (1,..,|r_p|],\\
&\varepsilon_{v,w} = \mathbf{u}(r_o, q) = \mathbf{u}(r_p, x),\\
&\varepsilon_{a,v} = \mathbf{u}(r_o, q-1), \varepsilon_{b,v} = \mathbf{u}(r_p, x-1):\\
&((f^{q}_{k,m}.\phi + m \times f_k.t + \delta \leq f^{x-1}_{l,n}.\phi + n \times f_l.t + \varepsilon_{b,v}.d(f_l.c))\vee\\
&(f^{x}_{l,n}.\phi + n \times f_l.t + \delta \leq f^{q-1}_{k,m}.\phi + m \times f_k.t + \varepsilon_{a,v}.d(f_k.c)))\vee\\
&(f_k.p \neq f_l.p).
\end{aligned} \tag{4.6}$$

The *Flow Deadline constraint* defined in Eq. (4.7) 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.

$$\begin{aligned}
&\forall \gamma_i \in \Gamma, \forall f_j \in \gamma_i.\mathcal{F}, \forall m \in [1,..,|f_j|], r_n = \mathbf{z}(f_j),\\
&\varepsilon_{a,b} = \mathbf{u}(r_n, 1), \varepsilon_{y,z} = \mathbf{u}(r_n, |r_n|):\\
&f^{1}_{j,m}.\phi + f_j.d \geq f^{|r_n|}_{j,m}.\phi + \frac{f_j.c}{\varepsilon_{y,z}.s}.
\end{aligned} \tag{4.7}$$

Task Constraints: We define three constraints for task scheduling on cores of FNs in the architecture model which relates the domain of the CP variables. A feasible solution meets all the constraints.

The *Core utilization constraint* avoids over utilization of cores. The constraint is defined in Eq. (4.8) where the utilization of all tasks and the server mapped to the same core is calculated.

$$\begin{aligned}
&\forall N_i \in \mathcal{N}, \forall \mathcal{C}_j \in N_i.C, \mathcal{T} = \{\tau_k | \mathcal{M}(\tau_k) = \mathcal{C}_j, \tau_k \in \gamma_l, \gamma_l \in \Gamma\} : \\
&\frac{D_{\mathcal{C}_j}.c}{D_{\mathcal{C}_j}.t} + \sum^{\tau \in \mathcal{T}} (\frac{\tau.c}{\tau.t}) \leq 1
\end{aligned} \tag{4.8}$$

The *Task Overlap constraint* imposes the restriction on the solution not to allow a core to run more than one job or server slice at a time. The constraint is defined in Eq. (4.4).

$$\begin{aligned}
&\forall \gamma_i, \gamma_j \in \Gamma, \forall \tau_k \in \gamma_i.\mathcal{T}, \forall \tau_l \in \gamma_j.\mathcal{T}, k \neq l, \\
&\forall m \in [1,..,|\tau_i.t|], \forall n \in [1,..,|\tau_i.t|], \\
&\mathcal{C}_o = \mathcal{M}(\tau_k) = \mathcal{M}(\tau_l), \forall y \in [1,...,H/D_{\mathcal{C}_o}.t] : \\
&((\tau_k^m.\phi + m \times \tau_k.t \geq \tau_l^n.\phi + n \times \tau_l.t + \tau_l.c) \vee \\
&(\tau_l^n.\phi + n \times \tau_l.t \geq \tau_k^m.\phi + m \times \tau_k.t + \tau_k.c)) \wedge \\
&((\tau_k^m.\phi + m \times \tau_k.t \geq D_{\mathcal{C}_o}^y.\phi + y \times D_{\mathcal{C}_o}.t + D_{\mathcal{C}_o}.c) \vee \\
&(D_{\mathcal{C}_o}^y.\phi + y \times D_{\mathcal{C}_o}.t \geq \tau_k^m.\phi + m \times \tau_k.t + \tau_k.c))
\end{aligned} \tag{4.9}$$

The *Deadline constraint* defined in Eq. (4.10) imposes the restriction that a job produces its outputs within its deadline.

$$\begin{aligned}
&\forall \gamma_i \in \Gamma, \forall \tau_j \in \gamma_i.\mathcal{T}, \forall m \in [1,..,|\tau_j.t|] : \\
&\tau_i^m.\phi + \tau_i.c \leq \tau_i.d.
\end{aligned} \tag{4.10}$$

Application Constraints: We define four constraints for handling the data dependency between tasks and flows of an application which is modeled with a DAG, see Sect. 4.2.3. To this end, we define four functions capturing the task and flow precedence.

The function $\mathcal{J}_\tau^{\mathcal{T}} : \gamma.\mathcal{T} \longrightarrow \gamma.\mathcal{T}$ takes an application task as the input and returns a set of tasks which have precedence over the input task within the same application. Similarly the function $\mathcal{J}_\tau^{\mathcal{F}} : \gamma.\mathcal{T} \longrightarrow \gamma.\mathcal{F}$ takes an application task as the input and returns a set of flows, the function $\mathcal{J}_f^{\mathcal{T}} : \gamma.\mathcal{F} \longrightarrow \gamma.\mathcal{T}$ takes an application flow as the input and returns a set of tasks, and the function $\mathcal{J}_f^{\mathcal{F}} : \gamma.\mathcal{F} \longrightarrow \gamma.\mathcal{F}$ takes an application flow as the input and return a set of flows, all within the same application.

The *Task Precedence over Task constraint* is defined in Eq. (4.11) and enforces that in an application and within a period, each task is executed after all the tasks determined by the function $\mathcal{J}_{\tau}^{\mathcal{T}}$ produce their outputs.

$$\begin{aligned}&\forall \gamma_i \in \Gamma, \forall \tau_j \in \gamma_i.\mathcal{T}, \forall \tau_k \in \mathcal{J}_{\tau}^{\mathcal{T}}(\tau_j),\\ &\forall l \in [1,..,|\tau_j.t|], \forall m \in [1,..,|\tau_k.t|]:\\ &\tau_j^l.\phi \geq \tau_k^m.\phi + \tau_k.c\end{aligned} \tag{4.11}$$

The flows' precedence over each task is regulated using the *Flow Precedence over Task constraint* defined in Eq. (4.12). The constraint avoids executing each task before all the flows decided by the function $\mathcal{J}_{\tau}^{\mathcal{F}}$ have arrived, within a period.

$$\begin{aligned}&\forall \gamma_i \in \Gamma, \forall \tau_j \in \gamma_i.\mathcal{T}, \forall f_k \in \mathcal{J}_{\tau}^{\mathcal{F}}(\tau_j),\\ &\forall l \in [1,..,|\tau_j.t|], \forall m \in [1,..,|f_k.t|],\\ &r_o = \mathbf{z}(f_k), \varepsilon_{a,b} = \mathbf{u}(r_o, |r_o|):\\ &\tau_j^l.\phi \geq f_{k,m}^{|r_o|}.\phi + \frac{f_k.c}{\varepsilon_{a,b}.s}\end{aligned} \tag{4.12}$$

Similarly, the *Task Precedence over Flow constraint* defined in Eq. (4.13) regulates the precedence of tasks determined by the function $\mathcal{J}_{f}^{\mathcal{T}}$ over each flow. Upon the output production of all determined tasks, the flow is scheduled for transmission.

$$\begin{aligned}&\forall \gamma_i \in \Gamma, \forall f_j \in \gamma_i.\mathcal{F}, \forall \tau_k \in \mathcal{J}_{f}^{\mathcal{T}}(f_j),\\ &\forall l \in [1,..,|f_j.t|], \forall m \in [1,..,|\tau_k.t|],\\ &r_o = \mathbf{z}(f_j), \varepsilon_{a,b} = \mathbf{u}(r_o, 1):\\ &f_{j,l}^1.\phi \geq \tau_k^m.\phi + \tau_k.c\end{aligned} \tag{4.13}$$

Eq. (4.14) defines the *Flow Precedence over Flow constraint* which regulates the transmission of a flow that is data dependent to a set of flows determined by the function $\mathcal{J}_f^{\mathcal{F}}$.

$$\begin{aligned}
&\forall \gamma_i \in \Gamma, \forall f_j \in \gamma_i.\mathcal{F}, \forall f_k \in \mathcal{J}_f^{\mathcal{F}}(f_j),\\
&\forall l \in [1,..,|f_j.t|], \forall m \in [1,..,|f_j.t|],\\
&r_n = \mathbf{z}(f_j), r_o = \mathbf{z}(f_k),\\
&\varepsilon_{a,b} = \mathbf{u}(r_n, 1), \varepsilon_{v,w} = \mathbf{u}(r_o, |r_o|):\\
&f_{j,l}^{1}.\phi \geq f_{k,m}^{|r_o|}.\phi + \frac{f_k.c}{\varepsilon_{v,w}.s}
\end{aligned} \tag{4.14}$$

4.4.3 Objective function

The CP solver propagates the constraints all over the search space 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 found.

We define the objective function Ω in Eq.(4.15). The terms Θ_1 and Θ_2 in the equation capture the analytical QoC and the extensibility, respectively. The weight β controls the trade-off between QoC and extensibility towards finding a solution with either a better QoC or a shorter response time for Fog applications. The weight β is between 0 to 2, where a larger β drives the search towards response time optimized solutions.

$$\Omega = \Theta_1 + \beta \times \Theta_2^{-1} \tag{4.15}$$

Analytical QoC CP model: In this work, we are interested in finding the solutions which have better QoC (see [CPBM19] for more information on the QoC and see [BCP20, BP21a] for more information on the schedule optimization for the QoC). Since calculating the QoC needs a simulation of the control application's behavior and the integration of QoC calculation tools such as JitterTime in the CP model are not feasible due to large runtimes, we have adapted the analytical QoC model proposed in [BP21a] and integrated it into our CP model.

The model formulates the QoC as: (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, called task execution interval, and (iii) minimum jitter for the task execution interval.

The analytical QoC CP model is formulated in Eq. (4.16), where γ_i is a critical control application and the terms θ_1 captures the input flow delay, θ_2 the output flow delay, θ_3 models the input flow jitter, θ_4 the output flow jitter, and θ_5 the task execution interval jitter. The range of all the θ terms is from 0 for no delay/jitter to 1 for a delay/jitter equal to the control application's period. The delay vs. jitter trade-off is controlled by the weight β_1 which can direct the search towards either optimized delay or optimized jitter, concerning the type of the control applications. A larger β_1 value drives the search towards smaller jitter. As proposed in [BP21a], the β_1 value is determined using JitterTime.

$$\begin{aligned}
&\forall f_j \in \mathcal{J}_t^f(\tau_1), \forall f_k \in \gamma_i.\mathcal{F} \setminus \mathcal{J}_t^f(\tau_1), \tau_1 \in \gamma_i.\mathcal{T}\\
&\forall m,q \in [1,..,|f_j|], \forall n,u \in [1,..,|f_k|],\\
&r_o = \mathbf{z}(f_j), r_p = \mathbf{z}(f_k):\\
&\theta_1 = \sum \frac{f_{j,m}^{|r_o|}.\phi}{f_j.t} \quad \theta_2 = \sum \frac{f_k.t - f_{k,n}^1.\phi}{f_k.t}\\
&\theta_3 = \sum \frac{|f_{j,m}^{|r_o|}.\phi - f_{j,q}^{|r_o|}.\phi + (m-q) \times f_j.t|}{f_j.t}\\
&\theta_4 = \sum \frac{|f_{k,n}^1.\phi - f_{k,u}^1.\phi + (n-u) \times f_k.t|}{f_k.t}\\
&\theta_5 =\\
&\sum \frac{|f_{k,m}^{|r_p|}.\phi - f_{k,q}^{|r_p|}.\phi + f_{j,q}^1.\phi - f_{j,m}^1.\phi + (m-q) \times f_j.t|}{f_j.t}\\
&\Theta_1 = \theta_1 + \theta_2 + \beta_1 \times (\theta_3 + \theta_4 + \theta_5)
\end{aligned} \tag{4.16}$$

Analytical extensibility CP model: This paper assumes using periodic slacks in the static task schedule for hosting Fog tasks and future critical control tasks; and periodic windows in queue gates for sending Fog flows and future critical control flows. We are interested in finding a distribution of the slacks and windows that provides a shorter response time for Fog applications and host a larger number of critical control applications. There are different techniques in the literature for analysis and determining such a distribution of the slacks and windows [WCP$^+$05, ZYS$^+$09, Alm03, LB03].

The typical technique is to first, calculate the required capacity, and then, determine the slack and window distribution that uses the least system resources and is able to provide the required capacity, at any time [BLAC05]. A common method for the calculation of the required capacity is the "submission load technique" [ABR$^+$93]. Thus, we define a general *Load function* $\mathcal{L}(t)$ in Eq. (4.17) for the required capacity calculation at time t. In Eq. (4.17), the function $u(t-a)$ is a delayed unit step function, n is the total number of tasks and flows in an application, j_k is the arrival time of the tasks and the flows, and c_k is the required capacity, i.e., the task workload and the message transmission time.

$$\mathcal{L}(t) = \sum_{i=1}^{n} \sum_{k=1}^{i} u(t - j_k) \times c_k \tag{4.17}$$

There are also different techniques for calculating the reserved resources in the real-time theory such as "server characteristic function" proposed in [LB03], "server supply function" proposed in [FM02], and "availability function" proposed in [Alm03]. We define the "Availability function" $\mathcal{A}(t)$ in Eq.(4.18), where the function $r(t-a)$ is a delayed unit ramp function, $S.t$ is the slack/window period, $S.c$ is the slack/window capacity, and the $S^i.\phi$ is the offset of the i^{th} slack/window slice.

$$\begin{gathered}\mathcal{A}(t) = \sum_{i=1}^{\lceil \frac{t}{S.t} \rceil} (r(t-\alpha_1) - r(t-\alpha_2)) \\ \alpha_1 = j \times S.t + S^i.\phi \quad \alpha_2 = \alpha_1 + S.c\end{gathered} \tag{4.18}$$

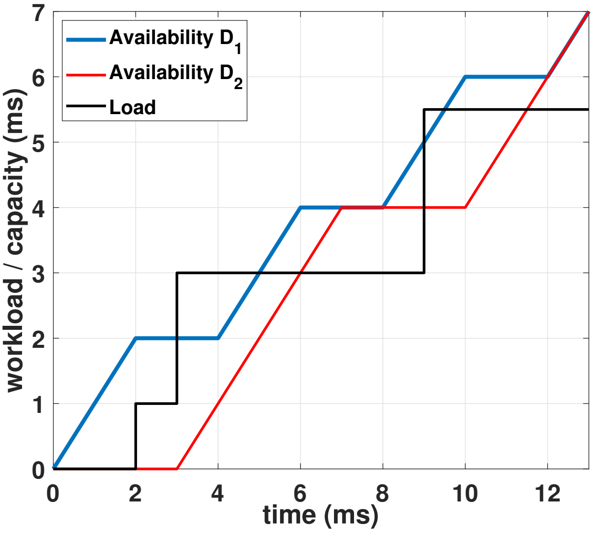


Figure 4.7: Example server availability and load: The server D_1 (blue curve) provides a shorter response time comparing to the server D_2 (red curve) for the known application load (black curve).

Once the load function and the server availability function are known, the response time *Res* can be calculated using Eq. (4.19) as proposed in [Alm03]. The equation Eq. (4.19) can be solved using iterative procedure starting at $t = 0$ until Res^{n+1} and Res^{n} converge. As mentioned, for Fog applications we focus on finding the solutions that provide a shorter response time, and for future critical control applications solutions that can host a larger number of applications without missing their deadlines. Such an analysis can also be used for future critical control applications as long as the determined response time is smaller than the deadlines.

$$Res = \mathcal{A}^{inv}(\mathcal{L}(Res)) \tag{4.19}$$

However, the presented analysis and design approach can not be employed at design-time since a prior knowledge on Fog applications and future critical control applications is not available (they arrive at runtime). To this end, we use the idea of the availability function in Eq. (4.18) and the extensibility metric presented in [BKA^{+}20] to define an analytic extensibility CP model in Eq. (4.20), where $S.t$ is the slack/window period, $S.c$ is the slack/window capacity, and the $S^{i}.\phi$ is the offset of the i^{th} slack/window slice. This analytic model calculates the accumulated reserved resources within a hyperperiod, capturing the distribution of the slack/window slices.

$$\begin{aligned}\Theta_2 &= \frac{\mathbf{n} \times S.c}{2} \times ((\mathbf{n}+1) \times S.t - S.c) \\ &- \sum_{i=1}^{n} S.c \times S^{j}.\phi, \quad \mathbf{n} = \frac{H}{S.t}\end{aligned} \tag{4.20}$$

We give an example Fog task handling problem in Fig. 4.7 where the black curve is the load function, the blue curve is the server D_1 availability function, and the red curve is the server D_2 availability function. The load function is calculated for the application γ'_1 consisting of three tasks specified as $\tau'_1\langle 1,2\rangle$, $\tau'_2\langle 2,3\rangle$, and $\tau'_3\langle 2.5,9\rangle$. The servers $D_1\langle 2,4\rangle$ and $D_2\langle 4,7\rangle$ have 7 and 4 server slices, respectively. For all slices of server D_1 and server D_2, the offsets are $D_1^{j}.\phi = 0$ and $D_2^{j}.\phi = 3$ ms.

We used the function in Eq. (4.19) to calculate the server's response time for the given tasks. Although the server D_2 has a higher utilization (57% comparing to 50% of the server D_1), it provides a loner response time for all tasks which is 3.5 ms (comparing to 3 ms of the server D_1). Thus, the server D_1 is a better choice as our proposed extensibility CP model approves. Our proposed extensibility CP model in Eq. (4.20) calculates the extensibility values of 210 and 200 for servers D_1 and D_2, respectively. As the example shows, our proposed analytic extensibility CP model has successfully determined the best server configuration, i.e., that provides the shorter response time.

4.4.4 Search Strategy

In this work we used the Google OR-Tools [Goo21] as the CP solver which is configured 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 good quality solutions in a reasonable time.

We used the same metaheuristic strategy as presented in [BZP20, BP21a] based on a Tabu Search metaheuristic algorithm [BK05], 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 all set of variables in the CP model. In this strategy, once a task or flow is scheduled with the respective minimum objective value, it is treated as keep variables whose values should not be changed.

4.5 Evaluation

This section is structured as follows: we first describe our evaluation setup and the test cases in Sect. 4.5.1. Afterwards, we evaluate our proposed ECOS method for its ability to support future critical control applications (Sect. 4.5.2) and to accommodate Fog applications (Sect. 4.5.3). Finally, we evaluate the performance of ECOS on a realistic test case where we consider scenarios of upgrades with future critical control applications and migrations of Fog applications (Sect. 4.5.4).

4.5.1 Test Setup and Scenarios

Our proposed ECOS approach is implemented in C# using Google OR-Tools [Goo21] as the CP solver and it is run on a computer with an i9 CPU at 3.6 Ghz and 128 GB of RAM. We limited the search in CP solver for a duration of 15 to 120 minutes depending on the test case size and the scenario. We assumed that all links have a speed of 1 Gbit/s.

We have generated ten synthetic test cases with critical control applications that have progressively larger number of tasks and flows, whose details are given in Table 4.2. The columns 2, 3, 4, 5 and 6 in the table show the total number of critical control applications, tasks, flows, ESs, and SWs for the test cases, respectively. Each critical control application implements an LQG control function which is designed with Jitterbug [LC02] for controlling plants in the form of Eq. (4.21) where *a* and *b* are randomly

Table 4.2: Details of ten critical control (CC) synthetic test cases

#	Total no. of CC apps	Total no. of CC tasks	Total no. of CC flows	Total no. of ESs	Total no. of SWs	Mean util. of CC apps
1	5	17	11	3	1	34%
2	7	20	14	5	1	39%
3	10	24	18	7	2	42%
4	12	26	20	9	2	44%
5	15	28	25	11	2	48%
6	18	30	30	13	3	51%
7	20	34	34	15	3	53%
8	23	36	38	17	3	56%
9	27	40	45	19	4	59%
10	30	42	45	21	4	61%

chosen respectively from $[50, 100, 150]$ and $[100, 200, 300, 400]$ (see [BCP20] for more details).

$$G = \frac{a}{s^2 + b} \tag{4.21}$$

The critical control flows are generated randomly with message sizes to fit in single MTU-sized frames. The tasks are also generated with random WCETs. Tasks and flows of each critical control application have equal periods and deadlines in the form of 2^n ms, $n = \{0, 1, 2, 3, 4\}$. The column 7 in the table shows the mean utilization of critical control applications which is the average of tasks' CPU utilization and flows' bandwidth utilization.

4.5.2 Supporting future control applications

We were first interested to evaluate the ability of ECOS to generate configurations that allow updates, i.e., the addition of future critical control applications with no changes to the current configuration. For this purpose, we created for each test case in Table 4.2 a scenario where critical control applications have to be added in the future. As depicted in Table 4.3, columns 2 and 3 show for each scenario the total number of tasks/flows, and mean utilization of the applications, respectively.

Together with ECOS, we evaluated a version, called ECOS/E that does not optimize for

Table 4.3: Evaluation results on synthetic test cases

TC	Total no. of tasks / flows in FCCAs[1]	Mean util. of FCCAs[1]	Percentage of Supported FCCAs[1]		RT[2]of Fog application 1 Tasks: 16 Flows: 15		RT[2]of Fog application 2 Tasks: 21 Flows: 20		RT[2]of Fog application 3 Tasks: 35 Flows: 38	
#			ECOS	ECOS/E	ECOS	ECOS/E	ECOS	ECOS/E	ECOS	ECOS/E
1	36/58	57%	100%	78%	1.33	3.97	2.82	5.66	4.73	7.43
2	37/65	55%	100%	89%	1.42	1.91	2.75	4.92	6.74	9.56
3	30/20	50%	100%	96%	1.26	3.36	2.94	4.88	5.16	12.29
4	29/32	45%	100%	81%	2.17	3.27	3.64	5.65	5.18	7.34
5	33/40	44%	100%	96%	1.56	4.14	3.68	5.81	4.36	9.44
6	44/45	40%	100%	90%	1.47	2.96	3.14	4.17	4.96	5.82
7	37/35	39%	100%	83%	1.19	3.95	3.88	3.96	2.96	4.76
8	33/34	37%	100%	90%	2.18	2.93	3.14	5.84	2.84	9.64
9	25/28	31%	100%	98%	1.65	3.77	4.43	5.93	2.35	4.74
10	18/21	27%	100%	82%	2.37	3.79	4.82	5.97	2.85	6.68
Average			100%	87%	1.6	3.4	3.5	5.2	4.2	7.7

[1] Future Critical Control Application
[2] Response time in ms.

extensibility, i.e., it does not consider the extensibility term Θ_2 in the cost function Ω and instead optimizes only for QoC term Θ_1. For the configurations obtained with ECOS and ECOS/E on each test case, we used again ECOS to add the future critical control applications for the respective test case, considering that the configuration of the initial critical control applications cannot be changed. This is to avoid re-certification, as discussed in Sect. 4.1.

Since an extensible configuration aims to support future critical control applications, i.e., their tasks and flows have no missed deadlines, we show in columns 4 and 5 of Table 4.3 the percentage of supported tasks and flows in ECOS and ECOS/E, respectively. The conclusion is that ECOS can better support future critical control applications by considering the extensibility of the configurations during their optimization. The main message is that without considering the extensibility, i.e., using ECOS/E, none of the future upgrades can be performed; all values in column 5 under are below 100%, showing that the upgrades are not accommodated. As the results show, only ECOS can add all the future critical control application. In some cases, ECOS shows that it can support 23% more tasks and flows compared to ECOS/E. Thus, ECOS shows a good performance in supporting future control applications that avoids re-certification costs.

4.5.3 Response time analysis of Fog applications

We were also interested to evaluate the ability of ECOS to accommodate Fog applications. Hence, we created for each test case in Table 4.2 a scenario with three sets of Fog applications that will need to be hosted by the FCP at the same time with the critical control applications. Each set of Fog applications consists of different number of tasks and flows with random arrival times, flow sizes, and task workloads. Their details are presented in columns 6 to 11 of Table 4.3. We simulate their migration to the FNs of the FCP considering the migration mechanism from Sect. 4.2.1. Fog flows in Table 4.3 are Best Effort and are using the SP traffic type, as discussed in Sect. 4.2.2.

Similar to the previous section, we evaluated ECOS against ECOS/E, where extensibility has not been considered. Thus, as in the previous section, we used ECOS and ECOS/E to generate optimized configurations for each test case. As discussed in Sect. 4.3, we are interested to host Fog applications with the shortest response times. ECOS uses the extensibility terms Θ_2 in the cost function Ω to favor solutions that are extensible, i.e., they have well-dimensioned servers and windows for Fog applications.

Columns 6 to 11 in Table 4.3 report the mean response time of Fog applications under the respective configuration using the Eq. (4.19) in ms. As we can see from Table 4.3, by considering the extensibility when creating the configurations at design-time, we are able to accommodate Fog applications at runtime with reduced response times with an average 43%. In some test cases, such configurations generated by ECOS could reduce the response time by 3.71 ms (see test case 3) and improve the response time by 51% (see test case 1). When ECOS/E is used, we can see that the response times for most Fog applications are much larger. That is because although the total resources available to the Fog applications are the same in both the ECOS and ECOS/E configurations, the

Table 4.4: Fog-based pharmaceutical production line

App set	Apps	No of Tasks / flows	Bandwidth util.	CPU util.
Critical control app	Agitator	6 / 5	0.7%	8%
	Boiler	3 / 4	0.4%	7%
	Chiller	5 / 7	0.7%	10%
	Coater	9 / 13	1.1%	11%
	Pulverizer	8 / 9	0.6%	12%
	Tablet printer	14 / 15	1.2%	18%
	Conveyor	15 / 21	1.5%	21
Future critical control apps	Heat exchanger	6 / 8	0.9%	11%
	Sifters	9 / 10	1.1%	17%
	Cartoners	7 / 10	0.8%	9%
	Sealer	10 / 12	0.9	9%
Fog apps	Label check	5 / 8	NA[1]	NA[1]
	Machine maintenance	12 / 13	NA[1]	NA[1]
	Air quality check	15 / 12	NA[1]	NA[1]
	Stock check	8 / 4	NA[1]	NA[1]
	Safety check	8 / 14	NA[1]	NA[1]

[1] Not Applicable

deferrable servers and communication windows, which allocate the available resources to Fog applications, are not well dimensioned in the case of ECOS/E, and hence cannot be used at runtime. This shows the importance of configuring for extensibility in the Fog.

4.5.4 Extending with upgrades

The next question we were interested to answer is if this ability of ECOS to accommodate Fog applications holds when we add future critical control applications. This scenario is adopted form industrial settings where upgrades in the form of future critical control applications are introduced while the ability of the FCP to host Fog applications are still needed. To this end, we present a realistic test case where a set of critical control applications as shown in Table 4.4 are considered at design-time. The test case consists of 4 ESs and 6 SWs implementing a pharmaceutical production line. We use ECOS to generate an extensible FCP configuration at design-time for these applications. In this configuration the mean utilization of critical control applications is 47%, and the FCP is able to host Fog applications (see their details in Table 4.4) with a response time of 3.6 ms.

Once a set of future critical control applications with a mean utilization of 26% are introduced as an upgrade (see Table 4.4), we again use ECOS to accommodate this upgrade. In the upgraded configuration, the slack which has been diminished for hosting Fog applications, since part of it has been assigned to run the future critical control applications. When adding these future control applications, ECOS has also optimized the new upgraded configuration for extensibility. As we can see, ECOS was able to update the configuration to still accommodate the Fog applications in Table 4.4, at the cost of a small increase in their response times, i.e., 1.76 ms. Note that in practice these response times will be smaller, because the critical control tasks will not execute for their WCET. Instead, they will finish earlier than WCET, and as we discussed in Sect. 4.2.4, the servers will use at runtime the freed computational resources.

4.6 Related Work

There has been a lot of work in the literature for scheduling in mixed-criticality systems and several methods have been proposed, such as hierarchical scheduling [WZS$^+$14], task partitioning and scheduling [LRL09], container-based scheduling [YLL18], and mixed traffic schedulers [ZS00] for flows in particular. Nevertheless, we focus on designing for extensibility in mixed-criticality systems.

Designing for evolvability and extensibility in computer systems has also been addressed in the literature [GRS96, RLL94]. Extensibility is defined as an ability that enables future upgrades and changes in computer systems [RRW$^+$03]. These changes and upgrades can be implemented in different ways such as introducing new applications, migrating applications across processing nodes, and changes in required resources. An extensible system is designed in a way that accommodating changes and upgrades does not require re-designing and costly changes in the platform. Computer systems hosting safety-critical applications requires safety certification and any changes in the system configurations requires re-certification [KZ09]. However, in this paper we focus on a form of extensibility that is accommodating future applications, thus, an extensible mixed-criticality that also hosts safety-critical applications, does not require re-certification.

Pop et al. [PEPP04] propose an incremental scheduling algorithm for embedded systems which aims at facilitating hard real-time applications implemented as tasks and adding specific future tasks. The approach generates extensible schedules that can accommodate future tasks without disturbing the existing tasks. This is realized using the slack in the schedule.

The approach in [ZYS$^+$09] uses extensibility to target robust task scheduling in distributed systems. Any changes in the task requirements are considered an extension. The proposed robust scheduler is able to support these changes using the dimensionized slacks in the schedule. Zheng et al. [WCP$^+$05] propose a mathematical modeling approach for extensibility. In this work extensibility is in the form of accommodating future tasks. This approach determines a distribution of the slack among all tasks and targets all variations of future task sets. A benefit of this approach is that no prior specification of future tasks are required.

We porposed in [BKA$^+$20] an extensible scheduling algorithm for critical applications in an FCP. The proposed algorithm employs a heuristic approach that provides well-distributed slacks in the schedules of high-critical applications, which can be used for scheduling future critical applications. Yin et al. [YLL18] propose a task scheduling algorithm that targets extensibility by using resources in the Cloud. In this approach, tasks are scheduled either on FNs or in the Cloud using containers.

The work in the literature also addresses extensibility in the systems hosting mixed-criticality network messages. [WHL$^+$19] focuses on scheduling network messages in TSN networks in automotive where dynamic messages with less-criticality are needed to be scheduled with ones of high criticality (e.g., control engine); and optimizing the schedules to host more dynamic messages. Guo et al. [GGZ$^+$12] propose a method for mapping task to the control units of an automotive use case targeting extensibility. These tasks exchange messages in the automotive network.

The authors in [MWTP$^+$13] propose an approach which decides the mapping of applications to the processing elements, separates the mixed-criticality applications using partitioning, and schedules tasks and messages of the applications. This method aims for the extensible configuration which allows adding future applications. Similarly, the approach presented in [ZZZ+13], uses the worst-case response time analysis to decide on the task allocation, the signal to message mapping, and the assignment of priorities to tasks and messages. The approach optimizes the decisions for accommodating future applications.

4.7 Conclusions

Fog Computing is an enabler for Industry 4.0 where mixed-criticality applications are running on a shared computing platform. The Fog Computing Platform (FCP) has to separate such applications to protect high-criticality applications. The vision is to virtualize control applications and run them as software tasks and transmit them as network messages, while their safety, dependability, and performance are guaranteed. We proposed a design-time static configuration of the FCP. We considered on one hand, critical control applications that are isolated and handled with a static design-time FCP configuration, and on the other hand, Fog applications that are handled at runtime using pre-allocated resources. These pre-allocated resources are determined in the FCP configuration at design-time such that the FCP is able to accommodate upgrades in the form of future critical control applications, and dynamic changes in the form of Fog applications, minimizing their response times. The We implemented an optimization approach using Constraint Programming, a declarative programming paradigm where realistic constraints can be added. We have evaluated our approach on several test cases and demonstrated its ability to synthesize extensible configurations.

CHAPTER 5

Paper D: Electric Drives as Fog Nodes in a Fog Computing-based Industrial Use Case

Electric drives, which are a main component in industrial applications, control electric motors and record vital information about their respective industrial processes. The development of electric drives as Fog nodes within a fog computing platform (FCP) leads to new abilities such as programmability, analytics, and connectivity, increasing their value. In this study, we use the FORA FCP reference architecture to implement electric drives as Fog nodes, which we call "fogification". We designed our fogified drive architecture and its components using Architecture Analysis and Design Language (AADL). The design process was driven by the high-level requirements that we elicited. We used the fogified drive architecture to implement a self baggage drop system in which electric drives are the key components. We then evaluated the fog-based design using several key performance indicators, which reveal its advantages over the current drive architecture.

5.1 Introduction

Industry 4.0 is an industrial revolution via digitalization that affects all industries and sectors. Digitization increases productivity, flexibility, and product quality. Moreover, it reduces time-to-market and supports mass-customization [BBM+16]. When machines become connected with each other, sensors, and actuators, they form the Industrial Internet of Things (IIoT), which is expected to increase the global gross domestic product (GDP) value to USD 15 trillion by 2030 [DBNA15].

The convergence of Operational Technology (OT) and Information Technology (IT) drives this digital transformation [HGB14]. However, OT and IT use different computation and communication technologies [HGB14]. OT employs dedicated hardware and software to implement real-time and safety-critical applications that have stringent timing and dependability requirements. OT uses proprietary technologies, imposes information flow restrictions, and hence does not support the vision of Industry 4.0 [HGB14]. IT uses cloud computing, artificial intelligence (AI), and big data to bring flexibility, scalablity, reduced costs and faster development. However, IT is not suitable for industrial applications where non-functional properties related to timeliness and dependability must be guaranteed [GVCL14].

Because Industry 4.0 will only become a reality through the convergence of OT and IT [HGB14], a new paradigm called fog computing has been envisioned as an architectural means to realizing the OT/IT convergence [SP16]. Fog computing is a "system-level architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Things" [Ope21a]. A Fog Computing Platform (FCP) brings computation and storage resources closer to the edge of the network. An FCP is composed of several interconnected Fog Nodes (FNs), as shown in Fig. 5.1. Several types of FNs from powerful high-end FNs to low-end FNs with limited resources have been proposed by researchers [BMNZ14, PML+19] and have been developed by companies [TTT21, Neb21].

This study addresses industrial application areas in which *electric drives* [BN16] are present. Electric drives control electric motors and are ubiquitous in industrial installations in many domains such as the automotive, food and beverage, marine and offshore, hydraulics, refrigeration, and air conditioning domians. In this paper, we propose an extension to electric drives to enable their use as FNs within a fog computing architecture (we call this process *fogification*). Extending electric drives (which are naturally at the edge of the network, near the machines, sensors, actuators, and industrial data sources) with fog computing capabilities will guarantee effective collaboration among the devices, nodes, and Cloud [KSLP19].

By developing the electric drives as FNs, new features, such as programmability, analytics, and connectivity with customer clouds, are expected to increase their value [FOR21].

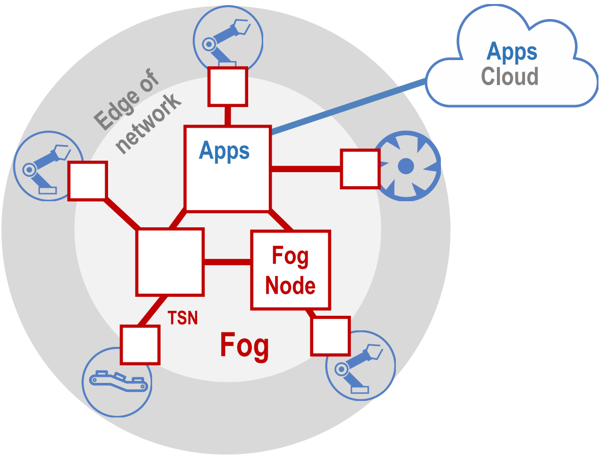


Figure 5.1: FCP: FNs (boxes) are placed at the edge of the network and connected via network (thick lines) with each other, equipment, and the Cloud. The applications run on FNs and in the Cloud.

Their increased functionality allows drives to assume a more significant role in industrial and domestic control systems by leveraging their ability to instrument as the data source, which can help bootstrap the data economy. The main direct business benefit comes from the ability to also instrument legacy systems using drives as the data source. Because electric drives run real-time software to control the speed, torque, and position of electrical motors that operate cooperatively with other devices to automate machinery, they produce data that carries vital information about the machinery they control. These data compromise a critical asset that is massive, often repetitive, and often must remain on-premises for privacy reasons. Hence, there is a need for drives to be capable of data analytics locally, at the edge.

Edge analytics will facilitate network off-loading and extend the Internet of Things (IoT) solution market. Digital services allow efficient service provisioning, improved uptime, and decreased overall costs. Correctly configured products and processes decrease energy consumption and improve quality. Open data ecosystems support innovations and new value-added services and will create long-term benefits for all ecosystem participants.

An initial limited investigation on extending electric drives with fog computing capabilities was carried out in [BDQ^{+}20]. In that study, we used the recently proposed FORA FCP reference architecture [PZB^{+}21] to design the electric drives as FNs using the Architecture Analysis & Design Language (AADL), which is an architecture description language from the domain of real-time embedded systems [FLV03].

AADL has been standardized by the Society of Automotive Engineers (SAE) and employs a component-oriented approach for modeling systems using both textual syntax and graphical notation with precise semantics [FLV03].

Furthermore, the AADL core language can be extended using user-defined properties and language annexes. This study builds on and extends existing annexes, such as the ARINC653 AADL annex [WMZ$^+$11], which defines virtual processors and virtual buses. AADL is supported by several tools for graphical modeling and analysis of embedded systems such as OSATE [T$^+$06], which is an open-source Eclipse-based framework consisting of a modeling environment and a set of plug-ins for validating and analyzing models. We also used OSATE for modeling, model checking, integration, and analysis.

The contributions of this paper are as follows: We propose a new design for electric drives as FNs using the FORA FCP reference architecture from [PZB$^+$21] and model the fogified drive architecture using AADL. Additionally, we implement a realistic industrial application that uses electric motors to drive conveyor belts using the proposed fogified drive architecture. We identify the fog-based drive requirements for driving the design process. Finally, we use the fog-based drives to develop a solution for our use case, and propose several Key Performance Indicators (KPIs), which are used to evaluate the fog-based solution. The evaluation states that the fog-based electric drives provide the FN functionalities as envisioned in [PZB$^+$21], as well as the typical drive functionality.

The remainder of this paper is structured as follows. We introduce electric motors, drives, the current architecture of drives, and the drive requirements in Section 5.2. We proposed our fogified drive architecture and its AADL model in Section 5.3. Section 5.4 evaluates the fog-based solution for our use case. We highlight the related work in Section 5.5 and conclude the paper in Section 5.6.

5.2 Electric Motors and Drives

We introduce electric motors and electric drives in Section 5.2.1 and Section 5.2.2, respectively, and describe an industrial setting in which motors and drives are used in Section 5.2.3. We present the current drive architecture and its AADL model in Section 5.2.4. Section 5.2.5 presents the drive requirements and KPIs.

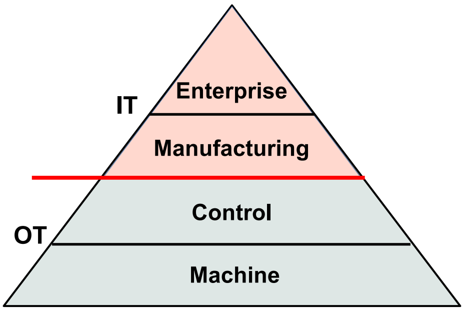


Figure 5.2: Automation pyramid.

5.2.1 Electric Motors

Electric motors are one of the main components in industrial settings in which machines are used for automation. As mentioned, they are used in many application areas, from building automation, energy systems, and industrial automation, mobile hydraulics.

An electric motor is an electromechanical machine that converts electricity into mechanical energy. The electric current is fed to the motor using a wire winding. This winding interacts with the motor's magnetic field, which applies torque to the motor shaft. Several classifications of electric motors have been introduced. For example, a well-known classification is based on the type of power source, Direct Current (DC) or Alternating Current (AC). Other classifications consider the internals of the electric motors or output motion [HD19].

Electric motors are capable of generating continuous rotation and are widely used in various areas, ranging from electric watches to ship propulsion. In an industrial setting in which electric motors are operating, a specific rotation scenario is assumed. For example, an electric motor is needed to generate a fixed speed rotation, e.g., 50 revolutions per minute (rpm) for 10 s, starting at a given time. The rotation may also have a specific speed, torque, and position, which can be controlled by altering the electric current, e.g., the voltage, amperage, and frequency (in AC).

5.2.2 Electric Drives

Electric drives, alternatively called *drives*, are used to alter the electric current's characteristics such as the frequency and voltage to control the motor's rotation for a required outcome, i.e., the desired speed, torque, and position [BN16]. Drives are designed to be general purposes, e.g., to control motors within a certain power range, or for specific

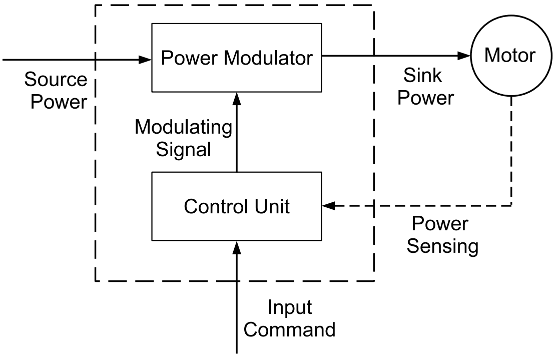


Figure 5.3: Basic block diagram of an electric drive.

purposes, e.g., to control an electric motor with specific requirements. An example of a specific purpose is the control of a centrifuge machine's motor for spinning very dense fluid, which needs very precise control. Because using drives along with electric motors is necessary in industrial settings and products, the market for drives is huge, and it is predicted that it will grow by USD 5.11 billion between 2019 and 2023 [Tec21]. Electric drives are a natural entry point for novel technologies that will bring significant business benefits, considering their role in the industry and their market value.

Figure 5.2 shows the so-called "automation pyramid" [Wil94], which captures the multiple levels of an industrial automation enterprise. The "machine level" consists of sensors and actuators, including electric motors, which are placed in the field or on the production floor. In contrast, the "control level" consists of industrial controllers such as Industrial Personal Computer (IPC) and Programmable Logic Controllers (PLCs), which control and manipulate the devices in the field. The controllers obtain their inputs from the machine level, e.g., sensors, switches, and Human–Machine Interfaces (HMIs), run a control algorithm to determine the desired outputs, and return the outputs to the actuator devices in the field. Although drives implement controllers to control the rotation of electric motors, they physically reside on the machine level and are classified as secondary control devices.

Drives are embedded cyber-physical systems that must meet real-time responses and reliability guarantees in order to meet the high dependability requirements of the application areas in which they are used. They run real-time applications and have access to detailed information about the electric motors they control and the industrial processes that are implemented.

The basic block diagram of a drive is given in Fig. 5.3, where the source power and command are the inputs and the sink power is the output. The drive has two internal units: (i) a power modulator unit and (ii) a control unit. The power modulator unit

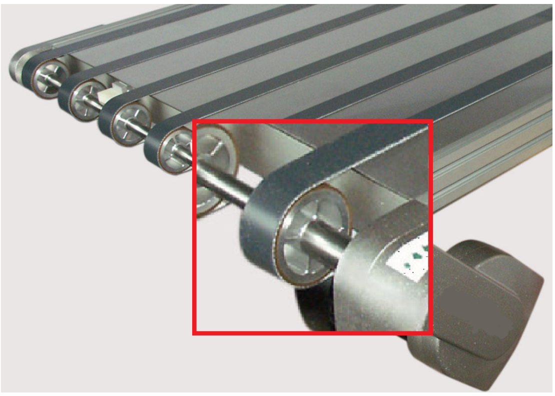


Figure 5.4: Conveyor belt.

modulates the source power of the modulating signal and returns the sink power. The modulating signal is generated by the control unit, which performs a resource-intensive computation based on the input command and sensed value of the sink power. The control unit may have safety features embedded such as a motor brake.

5.2.3 Example Industrial Setting

We provide an example industrial setting in Fig. 5.4, where an electric motor is used to control the conveyor belt speed. The conveyor belt machine is a carrying medium that uses belts rotating about two or more pulleys (the red box in the figure). The conveyor belt needs to move the load it carries with a specific speed profile determined by an industrial controller residing at the control level (Fig. 5.2), a PLC or an IPC.

Figure 5.5 shows a typical implementation consisting of a PLC, a sensor, a drive, and an electric motor. The sensor reads the load position and sends the data to the PLC, which determines the electric motor's current speed. The PLC sends the desired speed value to the drive. The drive controls the speed of the electric motor so that it is the desired value at the right time.

5.2.4 Baseline Drive Architecture

We first discuss a typical generic non-fog drive implementation [BD12, GRHMM$^+$15, GBFD, BM07], which we refer to as the "baseline architecture". We illustrate its design using AADL, as shown in Fig. 5.6a.

The drive takes as inputs (i) the source power, which is the main AC power line, and (ii) the command via a fieldbus interface that connects to a PLC, and outputs a sink power that operates an electric motor. The HMI is used as both the input and an output. As shown in Fig. 5.6a, the architecture consists of four components: operation component, communication component, control component, and power component. Each component uses dedicated hardware and software and has access to a shared bus, enabling it to be physically separated. Input-wise, all components are powered by the source power, but only the communication component has access to the input command. Output-wise, the power component returns the sink power, and the operation component has access to the HMI.

The **operation component** determines the operation mode of the drive. Figure 5.6b depicts the hardware, which has a CPU, RAM, and storage, and the software, which consists of real-time applications (Apps) that are running on a real-time operating system (OS). The operation component connects to the other components via the shared bus (Fig. 5.6a).

The applications running in the operation components (also shown in Fig. 5.6b) are as follows: (1) the mode control application (ModeControlApp), which engages and disengages the motor controller; (2) the management application (ManagmentApp), which configures the communication and controller parameters; and (3) the monitoring application (MonitoringApp), which implements safety functions. The ModeControlApp starts and stops the operation of the electric motor by engaging or disengaging the motor controller component based on the HMI or command inputs via the fieldBus interface. The ManagementApp sets the drive parameters such as the communication parameters via the HMI and motor control configurations such as the desired output (which is received from the communication component) via the shared bus. The MonitoringApp monitors the drive operation and engages safety functions when necessary.

The **communication component**, as shown in Fig. 5.6c, consists of the common hard-

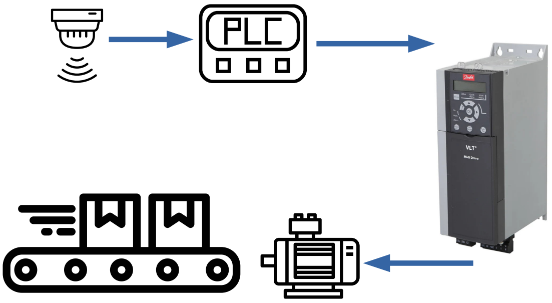
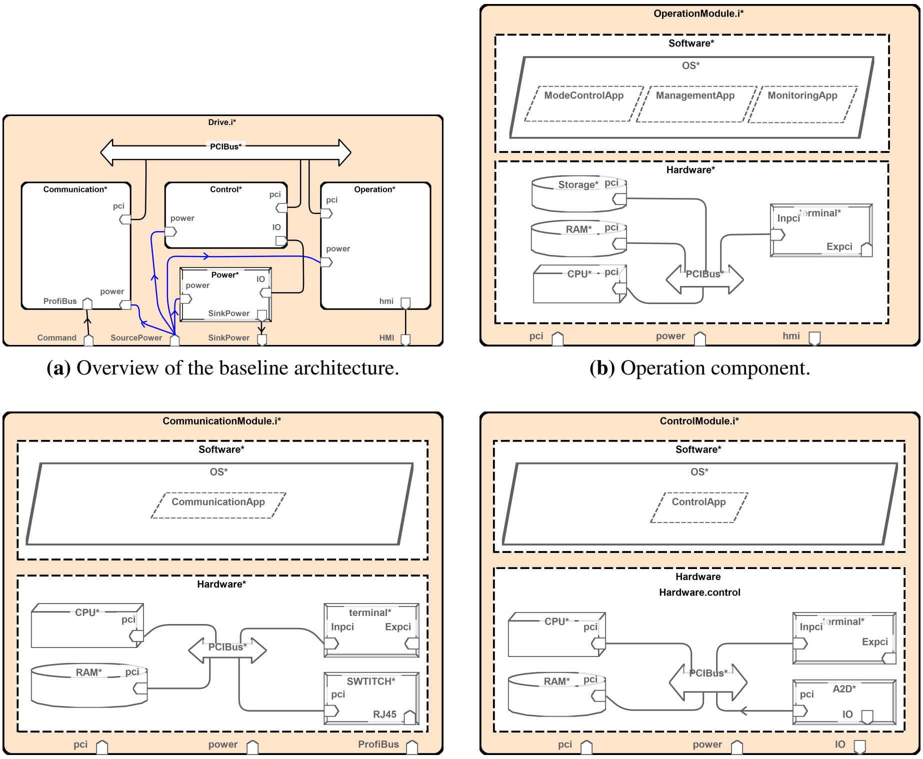


Figure 5.5: Typical implementation of a conveyor belt system.



(a) Overview of the baseline architecture.

(b) Operation component.

(c) Communication component.

(d) Motor control component.

Figure 5.6: AADL diagrams of the baseline architecture.

ware equipped with communication resources (the Switch box) to connect via the fieldBus interface with the ProfiNet/RT [Sie08] standard. The component's software consists of a real-time application (CommunicationApp) that handles the network protocol and is running on a real-time OS. The drive communicates with a PLC as the preliminary controller via a feildBus interface to obtain the desired motor output.

The **control component** is shown in Fig. 5.6d and consists of the hardware introduced earlier and software that implements a feedback control application (ControlApp) running on a real-time OS. The feedback control application is implemented according to the IEC 61131-3 standard [IEC03] function blocks. Once the application is engaged by the input from the operation component via the shared bus, it outputs a signal to the power component via the I/O interface. The signal is generated based on the configuration received via the shared bus from the operation component and the feedback received from the power component via the I/O interface.

Table 5.1: System-level requirements for Fog-based drives

#	**Requirement**	**Rationale**	**Realization in the fog-based architecture (Section 5.3.2)**
1	Drives shall be designed according to industrial standards.	To ensure drive compatibility with the industrial environment	IEC61800-based
2	Drives shall host mixed-criticality applications.	To virtualize critical and control applications and run IIoT applications envisioned in Industry 4.0	Virtualization and separation mechanisms
3	Drives shall perform accurate motor control.	To enable the virtualization of control applications to meet their non-functional properties (control performance)	Configuration to provide good control performance (20 ms response time)
4	Drives shall be configurable.	To enable the allocation of the necessary resources to mixed-criticality applications and to ensure portability and deployment of applications	Implemented via middleware
5	Drives shall have fault tolerant communication.	To ensure the connectivity and responsiveness of the drive	Via IEEE 802.1 TSN configuration [IEE21b]
6	Drives shall have a time-constrained communication interface.	To guarantee communication with bounded latency for critical applications exchanging messages	IEEE 802.1 TSN solution
7	Drives shall have Cloud access.	To enable deployment and communication for IIoT applications	Via Ethernet
8	Drives shall perform data analytics.	To avoid sending all data to the Cloud and to optimize resource management	Machine learning solutions
9	Drives shall enforce security policies.	To avoid malicious activity and define permissible user actions	Policies enforced via middleware
10	Drives shall be fault-tolerant.	To ensure fault-tolerant and high-integrity operation of safety-related applications	Fault detection, isolation, and recovery mechanisms

The **power component**, as depicted in Fig. 5.6a, takes the source power and the signal from the I/O interface as inputs and outputs the sink power, which operates the electric motor. This component is a power modulator that modulates the power line to the desired input reference. It also returns the current sink power setting via an I/O interface. The control component uses a feedback signal to control the sink power that operates the drive.

5.2.5 Requirements and KPIs

We elicited system-level requirements for designing a fogified drive architecture considering the vision of Industry 4.0. Table 5.1 lists the requirements and relevant rationale. The requirements specify that the fog-based drives should both have fog computing capabilities (for example, the ability to perform data analytics) and still deliver safety-related drive functions (for example, accurate motor control).

We have also defined several Key Performance Indicators (KPIs) that, should be used

Table 5.2: KPIs

#	Criteria	Motivation
1	Safety	Safety-critical industrial applications should be able to be hosted with no interference from less- criticality applications.
2	Security	Platform-level security services will be provided and the threat of successful attacks would be reduced.
3	Virtualized critical control performance	Using the configuration mechanisms, the FN should be able to configure its resources and the TSN switches according to the operational needs of the virtual control tasks such that the control performance can be guaranteed.
4	Hardware costs	Hardware spending should be reduced because the platform enables virtualization that converges various functions into one FN.
5	Data analytics	The FNs will implement the OPC unified architecture [MLD09] and connect directly to the equipment, sensors, and actuators. The data can then be analyzed locally using analytics applications at the edge. A subset of production data is then transferred securely to the Cloud for big data analytics.

to evaluate the solutions implemented using the fog-based drive architecture. Table 5.2 shows the KPIs and the relevant motivation.

5.3 Fog-based Electric Drives

In this section, we briefly present the FORA Fog Computing Platform (FCP) in Section 5.3.1, which was used to design our fogified drive architecture. We describe the proposed drive architecture and its AADL model in Section 5.3.2.

5.3.1 FORA FCP Reference Architecture

The FORA FCP reference architecture was introduced in [PZB^{+}21] to bring the fog computing paradigm to IIoT applications. The FCP consists of FNs connected to each other and to the machines through a deterministic communication solution, namely, IEEE 802.1 Time-Sensitive Networking (TSN) [IEE21b] (Fig. 5.1). The aim of TSN is to provide timing guarantees for demanding applications such as critical applications. It also guarantees bounded communication latency between nodes in the FCP. The key components of the FORA FCP are (1) deterministic virtualization, (2) middleware, and (3) mechanisms for resource management and orchestration [PZB^{+}21].

Because the FCP hosts mixed-criticality applications with differing requirements, an isolation mechanism is required to prevent low-criticality applications from interfering with high-criticality applications. In the FORA FCP, each FN utilizes a hypervisor that provides spatial and temporal isolation among the mixed-critically applications.

The FORA FCP uses middleware to support the implementation of distributed critical applications and non-critical applications such as IIoT applications that implement data analytics, updates, and regular checks. It also offers services for dependability, such as resource monitoring, safety and security monitoring, and machine learning services.

The FORA FCP employs resource management techniques for supply and demand alignment in an FCP and support configuration for optimizing resource utilization. With the resource management mechanisms, the available resources of each FN and the fog landscape are monitored at runtime, and services for the placement, deployment, and support of future applications and dynamic applications that may migrate across the FNs of the FCP are provided.

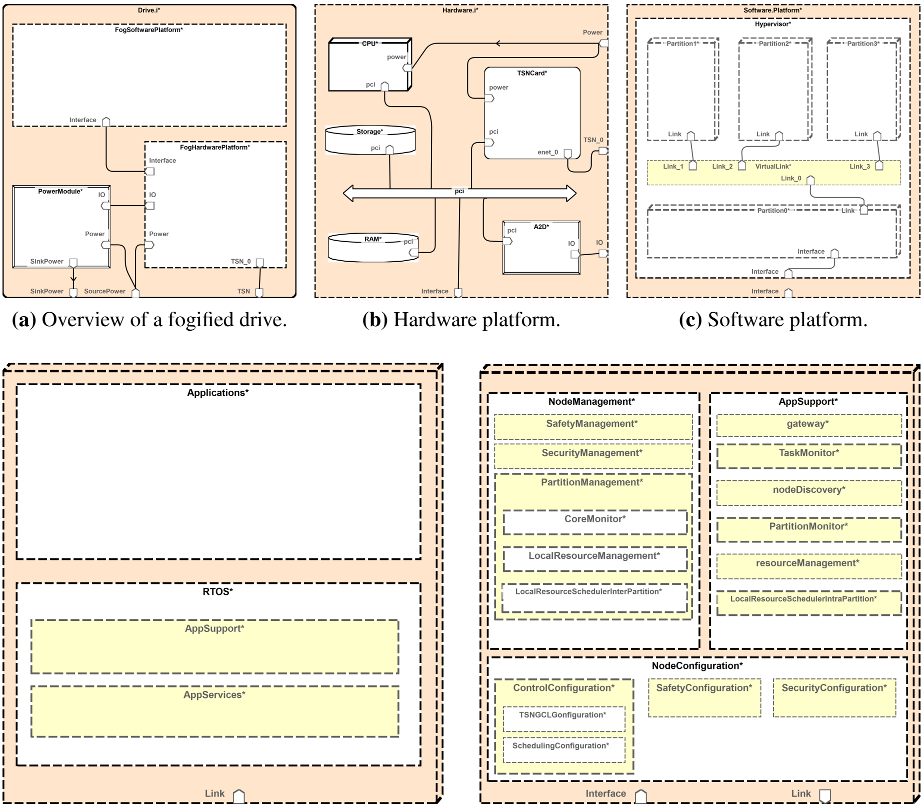
These services enable real-time decision making, security services, and resource prioritization. More information on the FORA FCP middleware is presented in [PZB$^+$21].

Resource management techniques align resource supply and demand in an FCP and support FCP configuration for optimizing resource utilization. With the resource management mechanisms, the available resource of each FN and the fog landscape is monitored at runtime, which allows the placement, deployment, and support of IIoT applications. More information on the resource management mechanisms for the FORA FCP is presented in [PZB$^+$21].

5.3.2 Fogified Drive Architecture

This section describes our proposed fogified drive architecture, which is based on the FORA FCP [PZB$^+$21]. We assume that using our proposed fogified architecture, drives are developed as FNs that are connected to each other and to the machines through TSN [IEE21b]. The fogified drives implement both the typical drive functionality (from the baseline architecture) and extra functionalities, as envisioned in the FORA FCP in an FN. We provide an overview of such a drive developed as an FN and modeled using AADL in Fig. 5.7a. The architecture consists of a hardware component, a software component, and a power modulator unit for operating electric motors. The fogified drive takes as inputs (1) the source power and (2) the network connection via the TSN interface. It outputs the sink power, and uses the network connection to send data.

The **hardware component** is depicted in Fig. 5.7b, which is equipped with a commercial of-the-shelf (COTS) multicore processor (CPU), RAM module, storage resources,



(a) Overview of a fogified drive. **(b)** Hardware platform. **(c)** Software platform.

(d) A partition. **(e)** Hypervisor.

Figure 5.7: AADL diagram of the fogified architecture.

an analog to digital converter module (A2D), and a network switch (TSNCard) for TSN capabilities. These resources are shared for running applications that sit on the software component. Similar to [PZB$^+$21], we consider a fog-based design that uses TSN because it supports mixed-criticality bounded-latency communication via multiple traffic types. For the scheduled traffic, which requires synchronized schedule tables, TSN employs a network-wide clock synchronization protocol, namely, IEEE 802.1AS [IEE21a] with sub-microsecond precision. The FN utilizes its advanced networking capabilities to interact with the environment, including sensors, actuators, other FNs, and remote Cloud facilities.

An example of a COTS multicore processor is the Intel Atom processor, which implements hardware virtualization extensions, such as Intel's VT-x and VT-d, second level address translation (SLAT), and single-root I/O virtualization (SR-IOV) [RS19].

Hardware virtualization extensions allow the hypervisor to host virtual machines with dedicated operating systems.

The **software component** has a software stack that consists of a hypervisor, middleware, partitions with dedicated operating systems, and an application layer, as shown in Fig. 5.7c. As mentioned in [PZB$^+$21], the software stack can take advantage of existing open-source software stacks for the edge, e.g., OpenStack [Fou21]. The middleware can use application layer protocols such as MQTT-SN [SCT13] or CoAP [SHB14] for higher-level component communication and TSN and OPC UA for lower-level component communication. Mixed-criticality applications that share the same hardware resources are separated into different virtual machines (partitions) enforced using hardware-supported virtualization [SVLN13] based on hypervisors such as PikeOS [KW07], ACRN [ACR20], or Xen [The21].

A hypervisor partitions its resources, such as processor cores and time, main memory, and I/O devices, to achieve strict temporal and spatial isolation of applications with mixed criticalities. The internals of our proposed hypervisor component are depicted in Fig. 5.7e, where we assume PikeOS [KW07] is used as the hypervisor. PikeOS can also make use of static partition schedules decided at design time to enforce temporal isolation. A static partition schedule consists of several partition slices where the partition is running based on a partition table that captures the start and end of partition slices (see [BCP20] for more information on partitions and partition slices).

The fogified drives can run all the drive applications from the baseline architecture, all of which are critical and modeled as periodic hard real-time tasks and messages [But11]. They can also run IoT applications that are non-critical, may migrate in and out of the drives, and implement tasks such as data analytics. Each IoT application was also implemented as tasks and messages. Our model assumes three types of applications— control, communication, and operation—which are assigned to separate partitions. The model for each partition is illustrated in Fig. 5.7d, where an OS runs the applications on the application layer. The OS runs the applications, and uses the AppServices and AppSupport for using shared resources and assuring dependability. The control partition has a soft-PLC OS, and the control application was implemented using the IEC 61131-3 standard [IEC03] function blocks. We also define a Quality of Control (QoC) for control applications that captures the control performance; see [CPBM19] for more information about the QoC. The communication partition has a real-time OS that runs applications for controlling network traffic, applying security mechanisms, handling application traffic, and deciding the TSN message schedule tables, called Gate Control Lists (GCLs). We assume that the operation partition has an OS to run different types of applications, including a machine learning application.

The **power modulator unit**, as depicted in Fig. 5.7a operates the same way as the power component in the baseline architecture. It takes the source power and the signal from the I/O interface as inputs and outputs the sink power, which operates the electric

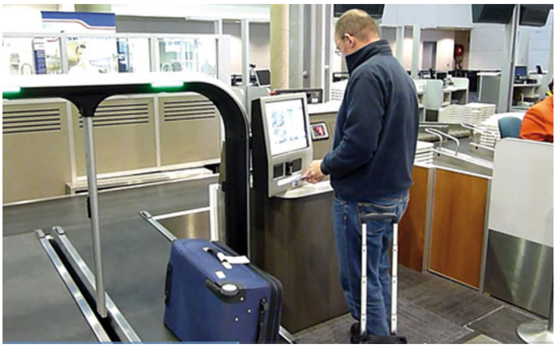


Figure 5.8: Self baggage drop system in Brisbane airport.

motor. The I/O interface is connected to the hardware platform where an analog to digital converter is accessible to control the partition that uses the I/O signal to operate the electric motors.

5.4 Evaluation

In this section, we first provide the details of the Use Case (UC), which we used for evaluation in Section 5.4.1. We evaluate the UC in Section 5.4.2 on several aspects using the KPIs in Table 5.2, discussing the suitability of the fog-based drive design for industrial applications.

5.4.1 UC Description

We used the proposed fogified drive to model a **self baggage drop system**. A self baggage drop system is a well-known and widely used machine in airports to automatically collect and distribute passenger baggage (Fig. 5.8). The machine is realized with several conveyor belts (see [BDQ$^+$20] for details of a conveyor belt), and it collects baggage from passengers and delivers the baggage to special vehicles (that carry the baggage to an airplane according to the destination). In this UC, we consider a typical machine, as depicted in Fig. 5.8, which is fed with baggage from every input location, weighs the baggage, reads the tag of the received baggage, and determines the destination of the baggage according to the tag by accessing a database. It then conveys the baggage toward the destination from one of the output locations, and then reports the machine usage and baggage delivery to the costumer's cloud.

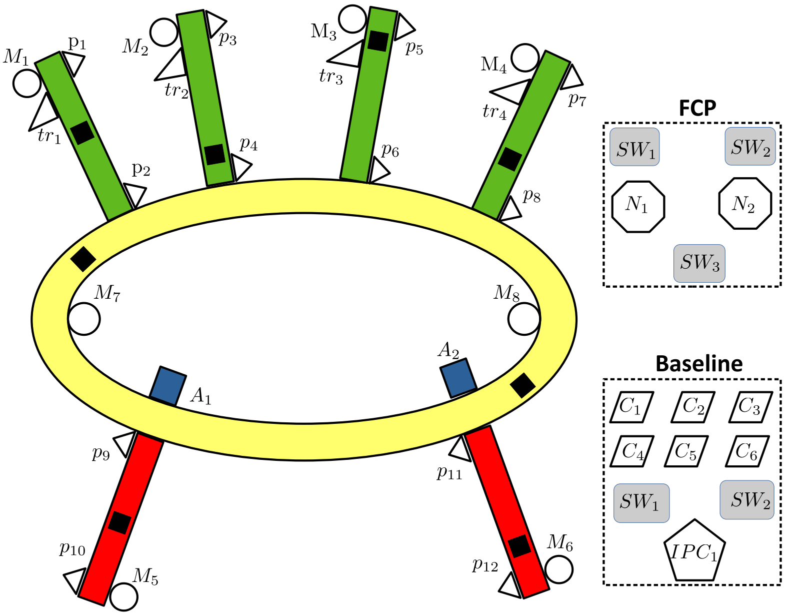


Figure 5.9: UC Schematics: The green belts are inputs, and the red belts represent the output. The round yellow belt is the main distribution belt. There are eight electric motors (M), 12 part present sensors (p), and four tag reader sensors (tr). The small black boxes represent the baggage. Network switches are indicated by SW in both the FCP and baseline architectures, whereas N denotes a fogified drive in the FCP architecture and C denotes a PLC in the baseline architecture. The baseline architecture has an IPC.

Figure 5.9 shows a schematic of the machine. As depicted in the figure, the machine has four input conveyor belts (green), one main distribution conveyor belt (yellow), two actuators (blue) that push baggage, and two output conveyor belts (red). Each input conveyor belt has an electric motor that drives a belt, two part-present sensors that sense the presence of a load, a tag reader for reading baggage tags, and a weight sensor for weighing the load. The main distribution conveyor belt has two electric motors for driving the belt and two actuators for pushing the loads. Each output conveyor belt also has an electric motor and two part-present sensors.

We consider two different implementations for the UC: (1) a baseline architecture and (2) our proposed fogified architecture. We also modeled the fogified implementation of the UC with AADL and show its diagram in Fig. 5.10. Table 5.3 summarizes the hardware equipment costs and number of units used in the two implementations of the UC.

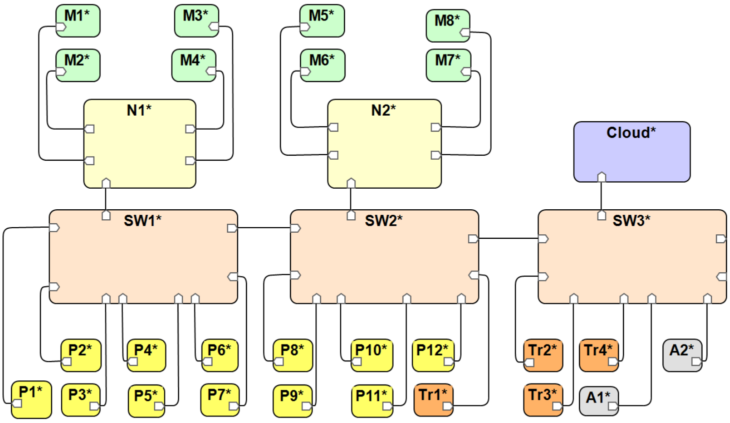


Figure 5.10: UC AADL model.

Drives in the fogified architecture cover the functionality of the baseline drives and can run all drive applications from the baseline architecture. We show example UC applications running on FN N_1 in Table 5.5, which shows the applications and relevant mapping of these applications to the processing elements of the baseline architecture.

5.4.2 Assessing the KPIs

In this section, we address each KPI separately and consider it as an evaluation criterion, using the evaluation method mentioned in Table. 5.4 to evaluate it on the UC. The results of the evaluation are as follows.

Safety: As mentioned, the FNs in the UC host mixed-criticality applications. High-criticality applications should be protected from low-criticality applications. This is achieved by spatial and temporal separation mechanisms implemented via the hypervisor. Consider the UC applications mapped to FN N_1 in Table 5.5, where column 1 shows the criticality of the application, ranging from 3 for the highest criticality to 0 for the lowest criticality.

The baseline architecture employs spatial separation that uses dedicated processing elements, i.e., drives, PLCs, and IPCs, to isolate applications of different criticalities. For example, high-criticality applications are assigned to run on PLCs, which guarantee deterministic execution, and the less-critical applications are assigned to be run on IPCs. Column 7 in Table 5.5 shows the mapping of the applications to processing elements.

Table 5.3: Summary of the UC hardware equipment

Equipment Name	**Unit Price (USD)**	**No. of units in the fogified Architecture**	**No. of units in the baseline Architecture**
Part present sensor	100	12	12
Electric motor	200	8	8
Tag reader	50	4	4
Weight sensor	50	4	4
Push actuator	100	2	2
Belt	50	7	7
PLC	500	0	6
IPC	1,000	0	1
Switch	100	3	2
Fogified drive	1,000	2	0
Electric drive	350	0	8

The same level of separation should be achieved in the fog-based solution. Hence, the fogified architecture uses a hypervisor and dependable middleware to achieve spatial partitioning. The hypervisor provides deterministic access to shared resources (spatial partitioning) and the temporal isolation of mixed-criticality applications via a static configuration table. The configuration of spatial and temporal isolation can be achieved using approaches such as the one proposed in [BCP20], which is used in the node configuration component (Fig. 5.7e). The configuration generates an optimized partition table for each FN, the mapping of applications to partitions based on their criticality levels, and the schedule tables for the critical tasks.

Table 5.4: Evaluation methods of KPIs

Criterion	**Evaluation**
#1	Provision of isolation via partitioning and evaluation of the applied overhead
#2	Protection of high-criticality applications and provision of authentication mechanisms for communication
#3	Optimization of the control performance for control applications
#4	Comparision of the hardware cost of a UC implementation using the baseline architecture and the fogified architecture
#5	Provision of a decentralized machine learning solution

Table 5.5: UC's applications running on the FN N_1

Criticality	Apps.	No. of tasks	No. of streams	Bandwidth utilization	Routing	Mapping (baseline)	App. core utilization	Total utilization w/o partitioning	Total utilization incl. partitioning
3	MC[1](γ_1)	2	1	0.2%	$p_1 \rightarrow SW_1 \rightarrow N_1$	D_1	15%	126%	139%
	MC[1](γ_2)	2	1	0.2%	$p_3 \rightarrow SW_1 \rightarrow N_1$	D_2	15%		
	MC[1](γ_3)	2	1	0.2%	$p_5 \rightarrow SW_1 \rightarrow N_1$	D_3	15%		
	MC[1](γ_4)	2	1	0.2%	$p_7 \rightarrow SW_1 \rightarrow N_1$	D_4	15%		
	MM[2](γ_9)	4	2	0.3%	$N_1 \rightarrow SW_1 \rightarrow SW_2 \rightarrow N_2$ $N_2 \rightarrow SW_2 \rightarrow SW_1 \rightarrow N_1$	C_1	12%		
	MM[2](γ_{10})	4	2	0.3%	$p_9 \rightarrow SW_2 \rightarrow SW_1 \rightarrow N_1$ $p_{11} \rightarrow SW_2 \rightarrow SW_1 \rightarrow N_1$	C_2	12%		
	MM[2](γ_{11})	4	2	0.3%	$N_1 \rightarrow SW_1 \rightarrow SW_2 \rightarrow SW_3 \rightarrow A_1$ $N_1 \rightarrow SW_1 \rightarrow SW_2 \rightarrow SW_3 \rightarrow A_2$	C_3	12%		
	SS[3](γ_{15})	2	0	0%	–	N/A	10%		
	SS[3](γ_{16})	2	0	0%	–	N/A	10%		
	SS[3](γ_{17})	2	0	0%	–	N/A	10%		
2	SR[4](γ_{21})	1	1	0.1%	$p_2 \rightarrow SW_1 \rightarrow N_1$	C_1	1.5%	18%	20%
	SR[4](γ_{22})	1	1	0.1%	$tr_1 \rightarrow SW_2 \rightarrow SW_1 \rightarrow N_1$	C_1	1.5%		
	SR[4](γ_{23})	1	1	0.1%	$p_4 \rightarrow SW_1 \rightarrow N_1$	C_2	1.5%		
	SR[4](γ_{24})	1	1	0.1%	$tr_2 \rightarrow SW_3 \rightarrow SW_2 \rightarrow SW_1 \rightarrow N_1$	C_2	1.5%		
	SR[4](γ_{25})	1	1	0.1%	$p_6 \rightarrow SW_1 \rightarrow N_1$	C_3	1.5%		
	SR[4](γ_{26})	1	1	0.1%	$tr_3 \rightarrow SW_3 \rightarrow SW_2 \rightarrow SW_1 \rightarrow N_1$	C_3	1.5%		
	DA[5](γ_{33})	4	3	0.5%	$N_1 \rightarrow SW_1 \rightarrow SW_2 \rightarrow SW_3 \rightarrow$ Cloud Cloud$\rightarrow SW_3 \rightarrow SW_2 \rightarrow SW_1 \rightarrow N_1$ $N_1 \rightarrow SW_1 \rightarrow SW_2 \rightarrow SW_3 \rightarrow$ Cloud	IPC_1	9%		
0	ML[6](γ_{34})	6	2	0.4%	$tr_1 \rightarrow SW_2 \rightarrow SW_1 \rightarrow N_1$ $N_1 \rightarrow SW_1 \rightarrow SW_2 \rightarrow SW_3 \rightarrow$ Cloud	N/A	8%	16%	17%
	ML[6](γ_{35})	6	2	0.4%	$tr_2 \rightarrow SW_3 \rightarrow SW_2 \rightarrow SW_1 \rightarrow N_1$ $N_1 \rightarrow SW_1 \rightarrow SW_2 \rightarrow SW_3 \rightarrow$ Cloud	N/A	8%		
	Sum							160%	175%

[1] Motor Control
[2] Machine Management
[3] Safety Service
[4] Sensor Reading
[5] Database Access
[6] Machine Learning

The advantage of virtualization and partitioning is that it reduces hardware costs, i.e., more equipment and mixed-criticality functionality can be hosted as software tasks on the FNs. However, we were interested in determining whether we could achieve a level of performance equal to that of the baseline architecture, that is, whether the virtualization would introduce much overhead. We evaluate this via a performance index defined as the number of processing element types and the overhead introduced by the separation mechanisms. As shown in Table 5.5, the applications are mapped to the FN N_1 in the fogified architecture, whereas on the baseline architecture, the processing elements are drives, PLCs, IPCs, and the Cloud; this result demonstrates that the performance on the fogified architecture is increased.

We evaluated the performance of our solution employed on the fogified architecture. As shown in Table 5.5, the example applications running on FN N_1 are separated using the proposed solution via partitioning. Our solution successfully determined the partitions and partition slices where the applications implemented as tasks are running. Moreover, it generated partition schedules that capture the start and end of partition slices.

The applications in Table 5.5 are implemented as tasks, and column 8 shows the utilization of each task when implemented on its respective core, i.e., the fraction of the 100% core utilization. Column 9 in the table shows the total utilization for each application without considering the overhead introduced by the partitioning-based virtualization; note that because the processors are dual-core, the total capacity of the two cores is 200%. Column 10 shows the total utilization when virtualization is used, accounting for the overheads. Three partitions were generated, and the core utilization of each partition increased by an average of 9% compared with the total utilization of applications without partitioning. In other words, the results show that to enable the safety-related separation of mixed-criticality applications, our proposed solution introduces only a 9% overhead on the total FN utilization.

Security: Security mechanisms are required to adequately protect the system against adversaries. A compromised system may allow the safety requirements to be violated. We briefly discuss security solutions that can be used in the UC's fogified architecture. These should all be deployed in parallel, as an instance of a defense-in-depth approach [Smi03]. The various mechanisms proposed here are summarized in Table 5.6, see [TDDFD20] for details of the mitigation.

As shown in the table, security attacks can be divided into two categories: execution-based and communication-based. The execution-based attacks target the configuration of the FN to interrupt the execution of the assigned applications. The FNs in the fogified architecture employ user access policies and apply configurations that are determined by the node configuration component. Partitioning introduces another protection mechanism because safety-critical applications are isolated in separate partitions.

Communication-based attacks affect the operation of a system by tampering with the network communication. Because the fog-based solution uses TSN, we focus here on the security vulnerabilities of TSN networks. TSN relies on clock synchronization for some of its mechanisms, e.g., the scheduled traffic type implemented with GCLs via 802.1Qbv. Hence, clock synchronization can be a security weakness for TSN [IW17, PBO19, XX11].

We now consider execution-based attacks. We assume that an attacker has control of

Table 5.6: Threats and their mitigation

Threat	**Mitigation**
Man-in-the-middle, impersonation	Confidential, authenticated com. channels
Attack impact	Service isolation (e.g., partitions)
Remote attacks	Firewalls, endpoint whitelisting
DoS	Redundant network topologies
TSN security	Isolation of the TSN protocol, per-stream filtering
Physical attacks	Hardware token for configuration changes
Detection	Security monitoring services

FN N_1 and tries to interrupt the execution of high-criticality applications. The malware, which we denote as task γ_{36}, runs on a partition with criticality level 0 (Table 5.5), and targets the intruption of the partition with criticality level 3, which runs motor control applications. The security configuration component implemented in the node configuration component (Fig. 5.7e) monitors the execution of tasks according to their precomputed schedule tables, identifies tasks with suspicious activity (e.g., unresponsive tasks or unauthorized access to resources), and performs predefined actions on the task (e.g., stopping the task and recovering the application).

Performance of virtualized control: In the baseline architecture, critical control applications are assigned to run on dedicated processing elements, i.e., PLCs, which are configured to meet the non-functional performance requirements of the control applications, such as deadlines and QoC.

The fogified architecture uses deterministic hypervisors to virtualize applications on FNs, similar to [RS19], where hypervisors provide deterministic access to shared resources via a static configuration table and provide spatial and temporal isolation of mixed-criticality applications via partitioning. Because critical control applications are virtualized and implemented as tasks, the configuration of the fogified architecture (e.g., task and communication scheduling) has an impact on the QoC of control applications [BCP19, BZP20].

The control functions are monolithic in the baseline architecture and do not exchange critical messages over the Profinet fieldbus. However, in the fog-based solution, critical control tasks exchange messages with each other as industrial "things", e.g., as sensors and actuators. Furthermore, the fogified architecture shares the same communication medium, i.e., TSN, for hard and soft real-time, non-critical, and best-effort communication. TSN has mechanisms to guarantee the timing requirements of critical streams, e.g., via scheduling enforced by GCLs [BZP20], but these must be properly configured.

The node configuration component determines the configuration needed to guarantee good control performance, measured via the QoC metric. The component uses a metaheuristic solution proposed in [BCP20] to optimize the hypervisor partition tables, map

Table 5.7: QoC of control applications running on FN N_1

App.	**IO Jitter**	**Max E2E delay (μs)**	**QoC**
γ_1	0	31	0.089
γ_2	0	29	0.081
γ_3	0	33	0.101
γ_4	0	32	0.096

the tasks to the processing cores of the multi-core processors of the FNs, assign the tasks to partitions, and schedule the tasks inside the partition tables, optimizing the QoC of control applications. The node management component also employs the constraint programming-based schedule synthesis strategy, which aims to maximize the QoC and satisfy the deadlines of real-time messages, as proposed in [BZP20] to schedule the traffic.

We evaluated the control performance of the proposed UC solution. We assume that the applications in Table 5.5 are running on FN N_1, which has a dual-core processor and exchanges network streams via TSN. Columns 3 to 6 in the table show the task and stream details, which are the total number of tasks, total number of streams, total bandwidth utilization of streams on the 1 Gbps link, and the the routing, respectively. We assume that applications γ_1 to γ_4 are motor control applications that control the speed of the electric motors. Each motor control application exchanges a message with a part-present sensor, calculates the control function similar to the function presented in [BCP19], and applies the respective output signal to electric motors.

Our proposed strategy for communication scheduling in TSN successfully scheduled all streams, i.e., none of the deadlines were missed, and optimized the schedules for the QoC. We also evaluated the performance of our proposed optimization strategy for task scheduling, and the proposed system successfully scheduled all the tasks and determined the task mapping to the cores.

The results show that all streams have zero jitter, which improves the QoC. We present the results in Table 5.7, where the I/O jitter, maximum end-to-end delay of the streams, and QoC values are reported using JitterTime [CPBM19], which simulates the behavior of the control application with respect to execution timing. We achieved a good control performance with an average QoC value of 0.092 using the objective defined in [BCP20].

Hardware cost: From a monetary perspective, the fogified architecture provides incentives concerning reducing the hardware cost. The common equipment cost for both the architectures C_{cost} of implementing the system shown in Fig. 5.9 is the sum of the equipment costs that are common for both, i.e., the first six items of Table 5.3. We compute the common equipment costs as C_{cost} = 3.750 (all values are in USD). The architecture-specific costs can be divided into baseline cost B_{cost} and fogified cost F_{cost}. Hence, the total costs of the baseline and fogified architectures are T^B_{cost} and T^F_{cost}, respectively. Therefore, $T^B_{cost} = B_{cost} + C_{cost}$ and $T^F_{cost} = F_{cost} + C_{cost}$. We computed the total costs for the baseline architecture and the fogified architecture as $T^B_{cost} = 10,750$ and $T^F_{cost} = 6,050$, respectively.

Furthermore, we consider scenarios in which the UC is updated with new features. For example, the UC employs an application that stops the conveyor belts when they do not carry a load to save energy. In the fogified architecture, this application is deployed on

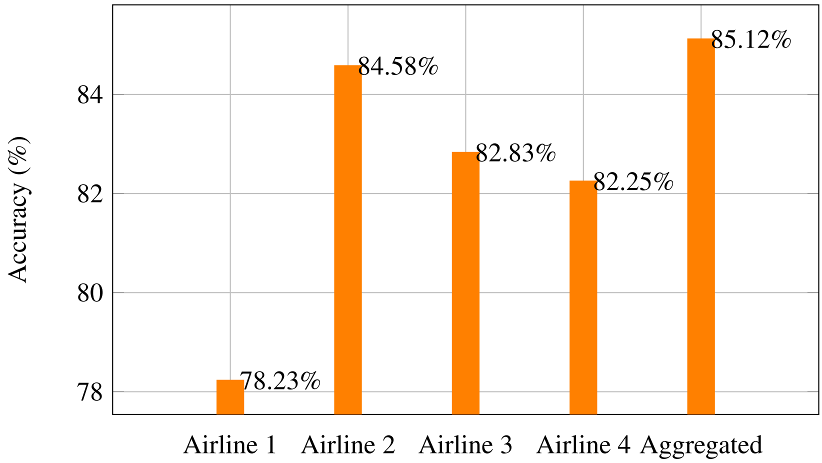


Figure 5.11: Distributed ML: four airlines jointly train a global model. The aggregated model outperforms a single entity after one-shot communication.

FNs via the Cloud connection. However, in the baseline architecture, the application is deployed on each processing element individually, increasing the management costs.

Data analytics: In a Cloud-based AI implementation, distributed participants upload their data to the Cloud, where the collected historical data is processed in a centralized fashion. The downsides of this approach are that (1) it is bandwidth- and time- consuming to upload this data and (2) it impairs privacy via potential data leakage. The solution is to utilize the edge device resources to carry out data analytics at the edge. A recent study [QH20] analyzed the minimal requirements of a reconstruction attack and provided insight into choosing the model size and architecture during the design of a Machine Learning (ML) algorithm. With the release of the General Data Protection Regulation (GDPR) [Pro21], data privacy protection has become a legal requirement.

We assume a scenario in which four airlines have decided to predict passenger satisfaction. As part of this survey, passenger baggage information must be processed. Implementing anML approach on the UC implemented with the baseline architecture is impossible because processing elements such as the PLCs and IPC do not provide sufficient resources, services, and connectivity to deploy such an application. Thus, a decentralized machine learning approach is implemented to run on the FNs of the fogified architecture.

The airlines aim to collaboratively train a predictor for passenger satisfaction. First, the airlines locally train their own models based on their own data. Second, they share the updated model with the Cloud based on their own data. Finally, the Cloud aggregates and updates the global model and sends it back to the airlines. The entire procedure can be repeated multiple times. The features we used in this experiment are passenger features such as gender and age, and flight features such as flight class, departure time, baggage weight, flight destination, number of bags, and baggage size. We cast all categorical features into numerical ones and normalized them to a range between zero and one during the preprocessing step. Here, we employ logistic regression for the binary predictor. Given $X \in \mathbb{R}^{N \times d}$, N is the total number of data and d is the dimension. Logistic regression maps the linear product of features to the range between zero and one using a sigmoid function, defined as $S(z) = \frac{1}{1+e^{-z}}$, where z is a linear combination of features defined as $z_i = \sum_j^d w_j x_{ij} \quad \forall i \in [1,N]$. In Fig. 5.11, we illustrate the advantage of the aggregation step using one-shot communication, where the aggregated accuracy of models created by individual airlines.

The implementation significantly decreases the upload bandwidth. For example, it may save $(B-1) \times 8d$ bytes for logistic regression model, where B is the batch size for one shot and, d is the data dimension. This will save $n \times (B-1) \times 8d$ for n-shot communications, and the advantage is more obvious with high-dimensional data (d is large). Moreover, the FN can reply to customer queries without a large delay, which is the main drawback of Cloud-based AI. The data remaining in the generation location significantly reduce the risk of privacy leakage.

5.5 Related work

Several research projects have addressed mixed-criticality applications that share multicore-based distributed architectures. The aims of the EMC2 European Project[1] are to provide efficient handling of mixed-criticality applications under real-time conditions, scalability and maximum flexibility, and full-scale deployment and management of integrated tool chains, throughout the entire life cycle. Research on FCP architectures has made progress in recent years [PML$^+$19, YHQL15]. For example, the European projects FORA [PZB$^+$21] and mF2C[2] focus on creating open-source, standards-compliant fog platforms using COTS hardware to execute hard real-time industrial control applications such as the electric drives discussed in this paper. Companies such as TTTech Computertechnik AG [TTT21] and Nebbiolo Technologies Inc. [Neb21] are pioneers in the field of commercializing the fog computing paradigm with market-ready products for industrial automation. Although design paradigms for the fog are still in their early stages, there are certain generic guidelines that are followed to ensure the isolation of tasks of varying criticality. In [PMN$^+$16], the authors describe an execution framework in which applications are isolated temporally on many-core processors.

[1] www.artemis-emc2.eu

[2] https://www.mf2c-project.eu

Safety certification as proof of guarantees for the proper execution of safety functions is needed for the FCP. Classical safety controller designs such as the simplex architecture [BCA^{+}09, Lui01] provide a switching mechanism between a high-performance but non-safety certified controller and a simple certified controller for safety functions. However, for complex systems such as the FCP, the simplex design is nonoptimal because of its switching latencies. Selicean et al. [TSP15b] proposed a method in which different safety-integrity levels (SILs) are assigned to the applications. In this method, applications with the same SIL are mapped to a single partition. Virtualization of control applications can then be realized by separating and scheduling the control tasks inside the partitions, similar to [BCP20]. The modification of hypervisors provides different degrees of separation. Modification of the Xen hypervisor [The21] to guarantee timing constraints was proposed by Masrur et al. [MDPC10]. The authors modified the hypervisor with a new scheduler based on a fixed-priority policy and a control loop to control the timing constraints of virtual machines. [DP19] addressed safety critical applications running in the fog and how the FCP must cater to these specific requirements.

One major research themes is resource management in the fog. In [HV19], the authors identified and classified the architectures, infrastructure, and underlying algorithms for managing resources in fog/edge computing. The authors of [PRGS18] proposed a list scheduling-based heuristics to solve this problem. The authors demonstrated the feasibility of reconfiguring the scheduled network at runtime for industrial applications within the fog. Reference [JPJ17] introduced a vulnerability-based method to quantify the security performance of communications on distributed systems. Fault-tolerant aspects were discussed in [JHZJ18], where the design problem is to minimize the schedule length and security vulnerability of the application, subject to given fault-tolerant constraints. A multi-objective optimization method was then proposed to find the best solutions. Reference [PGOP14] discussed potentially contradicting design constraints: real-time capability versus scalability. This paper suggested a design methodology and architecture as a step toward perfectly scalable real-time systems, i.e., systems with deterministic timing behavior and run-time reconfiguration.

Industry4.0 system architectures: Cyber-Physical System (CPS) in industrial infrastructures also deal with the combination of mechatronics, communication, and information technologies to control distributed physical processes and systems. They are designed as a network of interacting software and hardware devices and systems, many of them with a higher level of decision-making capability in two respects: autonomic with self-decision processes and collaborative with negotiation-based decision processes. Recently, a number of European projects have focused on flexible architectures for Industry 4.0-driven CPSs as well as distributed control systems (DCS). A common design goal for various reference architectures for Industry 4.0 is to introduce dynamic and flexible interaction among components [LCK16]. One of these initiatives is the German Industry 4.0 initiative, which specifies the Reference Architecture Model Industry 4.0 (RAMI 4.0) [DIN SPEC 91345].

We describe three architectural approaches detailed in the EU and German projects—PERFoRM, IMPROVE, and BaSys 4.0. Production harmonizEd Reconfguration of Flexible Robots and Machinery (PERFoRM) focuses on increasing flexibility and configurability in manufacturing. The primary goal is to transform existing systems into flexible and reconfigurable systems by providing an architecture with a common infrastructure for different industries.[3] The aim of the Innovative Modeling approaches for Production systems tO Raise Validatable Efficiency (IMPROVE) project is to develop a decision support system for tasks such as diagnosis and optimization. This is realized by the creation of a virtual factory that serves as a basis for model development and validation. Therefore, data from several systems in the plant need to be aggregated and integrated.[4] BaSys 4.0 stands for Basic System Industry 4.0, and it abstracts the overall production process and allows optimization prior to making the actual changes in the system. In addition, this system architecture provides real-time capabilities for critical process control functions [TWZ$^+$17].

5.6 Conclusions

In this study, we addressed electric drives, which are widely used in industrial applications. We proposed a way to re-engineer them as FNs, a process that we called fogification, based on the recently proposed FORA FCP reference architecture. We modeled and designed the fogified drive architecture using the AADL, capturing the main components and their interconnections. The design was driven by a set of requirements that we created that consider both the baseline functionality of the drives and their envisioned role as FNs. Fog-based drives are naturally located at the edge of the network, close to the machines, sensors, and actuators. Using the proposed architecture, we identified a solution for a self baggage drop system UC. We defined several KPIs for evaluating the suitability of our fog-based architecture for the UC. The evaluation results show improved performance, reduced hardware cost, and an increased analytics capability, without jeopardizing safety and security or the performance of virtualized critical control applications. As the evaluation shows, the fog-based drive architecture is a promising approach to implement the functionalities envisioned in Industry 4.0. In our future work, we will further integrate required technology components for the implementation of the use case.

[3] http://www.horizon2020-perform.eu

[4] http://www.improve-vfof.eu

Bibliography

[ABR+93] Neil Audsley, Alan Burns, Mike Richardson, Ken Tindell, and Andy J Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software engineering journal*, 8(5):284–292, 1993.

[ÅCES00] K. Årzén, A. Cervin, J. Eker, and L. Sha. An introduction to control and scheduling co-design. In *Proceedings of IEEE Conference on Decision and Control*, volume 5. IEEE, 2000.

[ACR20] ACRN. Official Website of the Project ACRN™. `http://projectacrn.org/`, 2019 (accessed May 1, 2020).

[AHG20] Anna Arestova, Kai-Steffen Jens Hielscher, and Reinhard German. Design of a hybrid genetic algorithm for time-sensitive networking. In *Measurement, Modelling and Evaluation of Computing Systems*. Springer International Publishing, 2020.

[AHM18] A. A. Atallah, G. B. Hamad, and O. A. Mohamed. Fault-Resilient Topology Planning and Traffic Configuration for IEEE 802.1Qbv TSN Networks. In *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design*, pages 151–156, 2018.

[Alm03] Luís Almeida. Response time analysis and server design for hierarchical scheduling. In *proceedings of the IEEE Real-Time Systems Symposium Work-in-Progress*. Citeseer, 2003.

[ALR+01] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, et al. *Fundamental concepts of dependability*. University of Newcastle upon Tyne, Computing Science, 2001.

[AS99] Tarek F Abdelzaher and Kang G Shin. Combined task and message scheduling in distributed real-time systems. *IEEE Transactions on parallel and distributed systems*, 10(11):1179–1191, 1999.

[ATD19] Cosmin Avasalcai, Christos Tsigkanos, and Schahram Dustdar. Decentralized resource auctioning for latency-sensitive edge computing. In *IEEE International Conference on Edge Computing*, pages 72–76, 2019.

[AW97] KJ Astrom and B Wittenmark. *Computer Controlled System*. Prentice-Hall, 1997.

[AYY15] Swati Agarwal, Shashank Yadav, and Arun Kumar Yadav. An architecture for elastic resource allocation in Fog Computing. *International Journal of Computer Science & Communication*, 6(2):201–207, 2015.

[BBB$^+$09] James Barhorst, Todd Belote, Pam Binns, Jon Hoffman, James Paunicka, Prakash Sarathy, John Scoredos, Peter Stanfill, Douglas Stuart, and Russell Urzi. A research agenda for mixed-criticality systems. *Cyber-Physical Systems Week*, 12, 2009.

[BBC17] P. Basquel, R. Burke, and P. Curran. Optimal closed-loop transfer functions for non-standard performance indices. In *Irish Signals and Systems Conference*, pages 1–6, June 2017.

[BBM$^+$16] Harald Bauer, Cornelius Baur, Detlev Mohr, Andreas Tschiesner, Thomas Weskamp, Knut Alicke, and D Wee. Industry 4.0 after the initial hype–where manufacturers are finding value and how they can best capture it. *McKinsey Digital*, 2016.

[BCA$^+$09] S. Bak, D. K. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha. The System-Level Simplex Architecture for Improved Real-Time Embedded System Safety. In *Proceeding of IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 99–107, 2009.

[BCP19] M. Barzegran, A. Cervin, and P. Pop. Towards Quality-of-Control-Aware Scheduling of Industrial Applications on Fog Computing Platforms. In *Proceeding of the ACM Workshop on Fog Computing and the IoT*, pages 1–5, 2019.

[BCP20] Mohammadreza Barzegaran, Anton Cervin, and Paul Pop. Performance optimization of control applications on fog computing platforms using scheduling and isolation. *IEEE Access*, 8:104085–104098, 2020.

[BD12] J Berthing and AS Danfoss. D5. 6 drive controller. *Artemis JU RECOMP Project*, 2012.

[BD13] Alan Burns and Robert Davis. Mixed criticality systems—a review. *Department of Computer Science, University of York, Tech. Rep*, pages 1–69, 2013.

[BD18] Alan Burns and Robert I Davis. A survey of research into mixed criticality systems. *ACM Computing Surveys*, 50(6):82, 2018.

[BDQ+20] Mohammadreza Barzegaran, Nitin Desai, Jia Qian, Koen Tange, Bahram Zarrin, Paul Pop, and Juha Kuusela. Fogification of electric drives: An industrial use case. In *Proceeding of the IEEE International Conference on Emerging Technologies and Factory Automation*, volume 1, pages 77–84, 2020.

[BDQP21] Mohammadreza Barzegaran, Nitin Desai, Jia Qian, and Paul Pop. Electric drives as fog nodes in a fog computing-based industrial use case. *Submitted to IET Journal of Engineering*, 2021.

[Bea04] Bernard C Beaudreau. *Mass production, the stock market crash, and the great depression: the macroeconomics of electrification*, volume 175. iUniverse, 2004.

[BFG+95] Edward Bensley, Lawrence Fisher, Mike Gates, James Houchens, Arkady Kanevsky, Soohee Kim, Peter Krupp, Alice Schafer, and Bhavani Thuraisingham. Evolvable real-time C3 systems. In *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems*, pages 153–166, 1995.

[BI07] Moris Behnam and Damir Isovic. Real-time control and scheduling co-design for efficient jitter handling. In *Proceeding of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 516–524, 2007.

[BK05] Edmund K Burke and Graham Kendall. *Search methodologies*. Springer, 2005.

[BKA+20] Mohammadreza Barzegaran, Vasileios Karagiannis, Cosmin Avasalcai, Paul Pop, Stefan Schulte, and Schahram Dustdar. Towards Extensibility-Aware Scheduling of Industrial Applications on Fog Nodes. In *Proceeding of the IEEE International Conference on Emerging Technologies and Factory Automation*, pages 67–75, 2020.

[BLAC05] Giorgio Buttazzo, Giuseppe Lipari, Luca Abeni, and Marco Caccamo. *Soft Real-Time Systems*. Springer, 2005.

[BM07] Jesper Berthing and Thomas Maier. Formalised Implementation of Safety Related HW/SW Architectures in Compliance with Functional Safety Requirements. In *Proceeding of the IET International Conference on System Safety*, pages 153–158, 2007.

[BM16] Dennis M Buede and William D Miller. *The engineering design of systems: models and methods*. John Wiley & Sons, 2016.

[BM20] Julian Bellendorf and Zoltán Ádám Mann. Classification of optimization problems in fog computing. *Future Gener. Comput. Syst.*, 107:158–176, 2020.

[BMNZ14] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. Fog computing: A platform for internet of things and analytics. In *Big Data and Internet of Things: A Roadmap for Smart Environments*, pages 169–186. Springer, 2014.

[BMZA12] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the Internet of Things. In *Proceedings of the ACM Workshop on Mobile Cloud Computing*, pages 13–16, 2012.

[BN16] Ion Boldea and Syed A Nasar. *Electric drives*. CRC press, 2016.

[BP21a] Mohammadreza Barzegaran and Paul Pop. Communication Scheduling for Control Performance in TSN-based Fog Computing Platforms. *IEEE Access*, 9:50782–50797, 2021.

[BP21b] Mohammadreza Barzegaran and Paul Pop. Extensibility-Aware Fog Computing Platform Configuration for Mixed-Criticality Applications. *Submitted to IEEE Transactions on Services Computing*, 2021.

[BPZ02] M.S. Branicky, S.M. Phillips, and Wei Zhang Wei Zhang. Scheduling and feedback co-design for networked control systems. *Proceedings of the IEEE Conference on Decision and Control*, 2(December):1211–1217, 2002.

[BRZ$^+$21] Mohammadreza Barzegaran, Niklas Reusch, Luxi Zhao, Silviu S. Craciunas, and Paul Pop. Real-Time Guarantees for Critical Traffic in IEEE 802.1Qbv TSN Networks with Unscheduled End-Systems. *arXiv preprint arXiv:2105.01641*, 2021.

[But11] Giorgio C Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*, volume 24. Springer Science & Business Media, 2011.

[BZP20] Mohammadreza Barzegaran, Bahram Zarrin, and Paul Pop. Quality-of-control-aware scheduling of communication in TSN-based fog computing platforms using constraint programming. In *Workshop on Fog Computing and the IoT*, pages 1–4. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[CBB17] Mung Chiang, Bharath Balasubramanian, and Flavio Bonomi. *Fog for 5G and IoT*, volume 288. Wiley Online Library, 2017.

[CEBÅ02] Anton Cervin, Johan Eker, Bo Bernhardsson, and Karl Erik Årzén. Feedback-feedforward scheduling of control tasks. *Real-Time Systems*, 23(1-2):25–53, 2002.

[Cer03] Anton Cervin. *Integrated Control and Real-Time Scheduling*. PhD thesis, Lund University, 2003.

[CJK16] Hyun Jun Cha, Woo Hyuk Jeong, and Jong Chan Kim. Control-Scheduling Codesign Exploiting Trade-Off between Task Periods and Deadlines. *Mobile Information Systems*, 2016, 2016.

[CLL18] Ching-Han Chen, Ming-Yi Lin, and Chung-Chi Liu. Edge computing gateway of the Industrial Internet of Things using multiple collaborative micro controllers. *IEEE Network*, 32(1):24–32, 2018.

[CO16] Silviu S Craciunas and Ramon Serna Oliver. Combined task-and network-level scheduling for distributed time-triggered systems. *Real-Time Systems*, 52(2):161–200, 2016.

[COCS16] Silviu S Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. Scheduling Real-Time Communication in IEEE 802.1 Qbv Time Sensitive Networks. In *Proceedings of the International Conference on Real-Time Networks and Systems*, pages 183–192, 2016.

[Coma] International Electrotechnical Commission. IEC 61784-2-12, Industrial communication networks-Profiles, Part 2: CPF 12 EtherCAT.

[Comb] International Electrotechnical Commission. IEC 61784-2-3, Industrial communication networks-Profiles, Part 2: CPF 3 PROFIBUS & PROFINET.

[Con05] FlexRay Consortium. Flexray communications system protocol specification version 2.1. `http://www.flexray.com`, 2005.

[COSS17] Silviu S Craciunas, Ramon Oliver Serna, and Wilfried Steiner. Formal scheduling constraints for time-sensitive networks. *arXiv preprint arXiv:1712.02246*, 2017.

[CPBM19] Anton Cervin, Paolo Pazzaglia, Mohammadreza Barzegaran, and Rouhollah Mahfouzi. Using JitterTime to analyze transient performance in adaptive and reconfigurable control systems. In *Proceeding of IEEE International Conference on Emerging Technologies and Factory Automation*, pages 1025–1032, 2019.

[CSB90] Houssine Chetto, Maryline Silly, and T Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Systems*, 2(3):181–194, 1990.

[CSE14] Silviu S. Craciunas, R. Serna Oliver, and V. Ecker. Optimal static scheduling of real-time tasks on distributed time-triggered networked systems. In *Proceedings of the IEEE Emerging Technology and Factory Automation*, pages 1–8, 2014.

[CSL18] Hoon Sung Chwa, Kang G. Shin, and Jinkyu Lee. Closing the Gap Between Stability and Schedulability: A New Task Model for Cyber-Physical Systems. In *in Proc. of IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 327–337, 2018.

[CSM16] T. Cruz, P. Simões, and E. Monteiro. Virtualizing programmable logic controllers: Toward a convergent approach. *IEEE Embedded Systems Letters*, 8(4):69–72, Dec 2016.

[CSOCS16] Silviu S. Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks. In *Proceedings of the International Conference on Real-Time Networks and Systems*, pages 183–192, 2016.

[DBNA15] Paul Daugherty, Prith Banerjee, Walid Negm, and Allan E Alter. Driving unconventional growth through the industrial internet of things. *Accenture Technology*, 2015.

[Dec05] J-D. Decotignie. Ethernet-based real-time and industrial communications. *Proceedings of the IEEE*, 93(6):1102–1117, 2005.

[DMP$^+$14] Erik Dahlman, Gunnar Mildh, Stefan Parkvall, Janne Peisa, Joachim Sachs, Yngve Selén, and Johan Sköld. 5g wireless access: requirements and realization. *IEEE Communications Magazine*, 52(12):42–47, 2014.

[DN16] Frank Dürr and Naresh Ganesh Nayak. No-wait packet scheduling for IEEE time-sensitive networks (TSN). In *Proceedings of the International Conference on Real-Time Networks and Systems*, pages 203–212, 2016.

[DP19] Nitin Desai and Sasikumar Punnekkat. Safety of Fog-Based Industrial Automation Systems. In *Proceeding of the ACM Workshop on Fog Computing and the IoT*, page 6–10, 2019.

[EES$^+$03] Jakob Engblom, Andreas Ermedahl, Mikael Sjödin, Jan Gustafsson, and Hans Hansson. Worst-case execution-time analysis for embedded real-time systems. *International Journal on Software Tools for Technology Transfer*, 4(4):437–455, 2003.

[EHÅ00] Johan Eker, Per Hagander, and Karl-Erik Årzén. A feedback scheduler for real-time controller tasks. *Control Engineering Practice*, 8(12):1369–1378, 2000.

[Ell66] F. J. Ellert. Performance indices for linear systems based on standard forms. In *Proceeding of the Symposium on Adaptive Processes*, pages 643–648, Oct 1966.

[Erb17] Gizem Erboz. How to define industry 4.0: main pillars of industry 4.0. *Managerial trends in the development of enterprises in globalization era*, pages 761–767, 2017.

[Eur16] Eurotech. Bridging the gap between operational technology and information technology. *Red Hat, Inc. White Paper. Available: https://bit.ly/3rHsfnO*, 2016.

[Eur21] European Telecommunications Standards Institute. Mobile edge computing framework and reference architecture. `http://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/01.01.01_60/gs_MEC003v010101p.pdf`, 2016 (accessed May 7, 2021).

[FGH06] Peter H Feiler, David P Gluch, and John J Hudak. The architecture analysis & design language (AADL): An introduction. Technical report, Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, 2006.

[FHC+19] Jonathan Falk, David Hellmanns, Ben Carabelli, Naresh Nayak, Frank Dürr, Stephan Kehrer, and Kurt Rothermel. NeSTiNg: Simulating IEEE time-sensitive networking (TSN) in OMNeT++. In *Proceeding of the International Conference on Networked Systems*, pages 1–8, 2019.

[FLV03] Peter H Feiler, Bruce Lewis, and Steve Vestal. The SAE avionics architecture description language (AADL) standard: A basis for model-based architecture-driven embedded systems engineering. Technical report, ARMY AVIATION AND MISSILE COMMAND REDSTONE ARSENAL AL, 2003.

[FM02] Xiang Feng and Aloysius K Mok. A model of hierarchical real-time virtual resources. In *Real-Time Systems Symposium*, pages 26–35, 2002.

[Fog21] Fog Computing and Networking Architecture Framework. IEEE 1934-2018 - IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing. `https://standards.ieee.org/standard/1934-2018.html`, 2018 (accessed April 1, 2021).

[FOR21] FORA. Fog Computing Platform: requirements and initial designs. http://www.fora-etn.eu/deliverables/, 2019 (accessed May 1, 2021).

[Fou21] Open Infrastructure Foundation. Open Source Edge Computing Architecture. https://www.openstack.org, 2012 (accessed April 1, 2021).

[FQ12] M. Fan and G. Quan. Harmonic semi-partitioned scheduling for fixed-priority real-time tasks on multi-core platform. In *Proceeding of the IEEE Design, Automation Test in Europe Conference Exhibition*, pages 503–508, 2012.

[GASR19] Mostafa Ghobaei-Arani, Alireza Souri, and Ali A Rahmanian. Resource management approaches in fog computing: a comprehensive review. *Journal of Grid Computing*, pages 1–42, 2019.

[GBFD] José Luis Gutiérrez, Jesper Berthing, David Fernández, and Javier Dıaz. Safety-critical platform model based on certification standards. *III Jornadas de Computación Empotrada, JCE*, 12.

[GGZ$^+$12] Liangpeng Guo, Arkadeb Ghosal, Haibo Zeng, Paolo Giusto, and Alberto Sangiovanni-Vincentelli. Methods and tools for calculating the flexibility of automotive hw/sw architectures. *SAE International Journal of Passenger Cars-Electronic and Electrical Systems*, 5(2012-01-0005):17–26, 2012.

[Gil16] Alasdair Gilchrist. Introducing industry 4.0. In *Industry 4.0*, pages 195–215. Springer, 2016.

[GITJ14] Omid Givehchi, Jahanzaib Imtiaz, Henning Trsek, and Juergen Jasperneite. Control-as-a-service from the cloud: A case study for using virtualized PLCs. In *Proceedings of the IEEE Workshop on Factory Communication Systems*, pages 1–4, May 2014.

[GJF12] Piotr Gaj, Jürgen Jasperneite, and Max Felser. Computer communication within industrial distributed environment—a survey. *IEEE Transactions on Industrial Informatics*, 9(1):182–189, 2012.

[GK17] Farid Golnaraghi and Benjamin C Kuo. *Automatic control systems*. McGraw-Hill Education, 2017.

[GK19] Mario Gleirscher and Stefan Kugele. Assurance of system safety: A survey of design and argument patterns. *arXiv preprint arXiv:1902.05537*, 2019.

[Glo21] Gartner IT Glossary. Operational technology. http://www.gartner.com/it-glossary/operational-technology-ot, 2015 (accessed March 3, 2021).

[GLSC17] Omid Givehchi, Klaus Landsdorf, Pieter Simoens, and Armando Walter Colombo. Interoperability for industrial cyber-physical systems: An approach for legacy systems. *IEEE Transactions on Industrial Informatics*, 13(6):3370–3378, 2017.

[GMS$^+$15] Thomas Goldschmidt, Mahesh Kumar Murugaiah, Christian Sonntag, Bastian Schlich, Sebastian Biallas, and Peter Weber. Cloud-based control: A multi-tenant, horizontally scalable soft-PLC. In *Proceedings of the IEEE International Conference on Cloud Computing*, pages 909–916, 2015.

[Goo21] Google. Google OR-Tools. `https://developers.google.com/optimization`, Accessed on May 1, 2021.

[GRHMM$^+$15] Jose Luis Gutiérrez-Rivas, Simon Holmbacka, Miguel Míndez-Macías, Wictor Lund, Sebastien Lafond, Johan Lilius, and Javier Díaz-Alonso. Safe motor controller in a mixed-critical environment with runtime updating capabilities. *Journal of Universal Computer Science*, 21(2):177–205, 2015.

[Gro21] IEEE 802.3 Ethernet Working Group. Standards for ethernet networks. `https://www.ieee802.org/3/`, 2021 (accessed April 1, 2021).

[GRS96] Michael Gagliardi, Ragunathan Rajkumar, and Lui Sha. Designing for evolvability: Building blocks for evolvable real-time systems. In *Proceedings of the IEEE Real-Time Technology and Applications*, pages 100–109, 1996.

[GVCL14] Marisol García-Valls, Tommaso Cucinotta, and Chenyang Lu. Challenges in real-time virtualization and predictable cloud computing. *Journal of Systems Architecture*, 60(9):726–740, 2014.

[GZPS17] Voica Gavrilut, Bahram Zarrin, Paul Pop, and Soheil Samii. Fault-tolerant topology and routing synthesis for IEEE Time-Sensitive Networking. In *Proceedings of the International Conference on Real-Time Networks and Systems*, pages 267–276, 2017.

[GZRP18] Voica Gavriluţ, Luxi Zhao, Michael L Raagaard, and Paul Pop. AVB-aware routing and scheduling of time-triggered traffic for TSN. *IEEE Access*, 6:75229–75243, 2018.

[HCÅ02] Dan Henriksson, Anton Cervin, and Karl-Erik Årzén. Truetime: Simulation of control loops under shared computer resources. *IFAC Proceedings Volumes*, 35(1):417–422, 2002.

[HD92] Kenneth Hoyme and Kevin Driscoll. Safebus. In *Proceedings of the IEEE/AIAA Digital Avionics Systems Conference*, pages 68–73, 1992.

[HD19] Austin Hughes and Bill Drury. *Electric motors and drives: fundamentals, types and applications*. Newnes, 2019.

[HDNQ17] Pengfei Hu, Sahraoui Dhelim, Huansheng Ning, and Tie Qiu. Survey on fog computing: architecture, key technologies, applications and open issues. *Journal of network and computer applications*, 98:27–42, 2017.

[Hea02] Steve Heath. *Embedded systems design*. Elsevier, 2002.

[HGB14] Derek R Harp and Bengt Gregory-Brown. IT/OT convergence bridging the divide. *NEX DEFENSE*, 2014.

[HV19] Cheol-Ho Hong and Blesson Varghese. Resource Management in Fog/Edge Computing: A Survey on Architectures, Infrastructure, and Algorithms. *ACM Computing Surveys*, 52(5), 2019.

[HZ19] Zhihong Huo and Zhixue Zhang. Scheduling and control co-design for networked wind energy conversion systems. *Global Energy Interconnection*, 2(4):328 – 335, 2019.

[IBM21] IBM Cloud Education. Hypervisors. `http://ibm.biz/hypervisors-guide`, 2021 (accessed April 1, 2021).

[IEC03] IEC. TIEC 61131–3 2nd Edition Programmable Controllers-Programming Languages. Technical report, IEC, 2003.

[IEE14] IEEE. 802.1Q-2014 - Bridges and Bridged Networks. `http://www.ieee802.org/1/pages/802.1Q.html`, 2014.

[IEE17] IEEE. 802.1ASrev—timing and synchronization for time-sensitive applications. `http://www.ieee802.org/1/pages/802.1AS-rev.html`, 2017.

[IEE21a] IEEE. Official Website of the 802.1 Audio Video Bridging Task Group. `https://www.ieee802.org/tsn/802-1as-rev`, 2013 (accessed April 1, 2021).

[IEE21b] IEEE. Official Website of the 802.1 Time-Sensitive Networking Task Group. `http://www.ieee802.org/1/pages/tsn.html`, 2016 (accessed May 1, 2021).

[IM94] David Isaac and Gail McConaughy. The role of architecture and evolutionary development in accommodating change. In *INCOSE International Symposium*, volume 4, pages 503–508. Wiley Online Library, 1994.

[Int10] International Electrotechnical Commission. IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems, 2010.

[IW17] Eyal Itkin and Avishai Wool. A security analysis and revised security extension for the precision time protocol. *IEEE Transactions on Dependable and Secure Computing*, 17(1):22–34, 2017.

[JAAH18] Mohammad Jbair, Bilal Ahmad, Mus'ab H. Ahmad, and Robert Harrison. Industrial cyber physical systems: A survey for control-engineering tools. In *IEEE Industrial Cyber-Physical Systems*, pages 270–276, 2018.

[JHZJ18] W. Jiang, H. Hu, J. Zhan, and K. Jiang. Work-in-Progress: Design of Security-Critical Distributed Real-Time Applications with Fault-Tolerant Constraint. In *Proceeding of the International Conference on Embedded Software*, pages 1–2, 2018.

[JPJ17] Wei Jiang, Paul Pop, and Ke Jiang. Design Optimization for Security- and Safety-Critical Distributed Real-Time Applications. *Microprocessors and Microsystems*, 52(C):401–415, 2017.

[JX07] P. Naghshtabrizi J. P. Hespanha and Y. Xu. A Survey of Recent Results in Networked Control Systems. *Proceedings of the IEEE*, 95(1):138–162, 2007.

[Kar19] Vasileios Karagiannis. Compute node communication in the fog: Survey and research challenges. In *ACM Workshop on Fog Computing and the IoT*, pages 1–5, 2019.

[KHS10] Andy Koronios, Abrar Haider, and Kristian Steenstrup. Information and operational technologies nexus for asset lifecycle management. In Dimitris Kiritsis, Christos Emmanouilidis, Andy Koronios, and Joseph Mathew, editors, *Engineering Asset Lifecycle Management*, pages 112–119, London, 2010. Springer London.

[KJ09] Mostafa Haghi Kashani and Mohsen Jahanshahi. Using simulated annealing for task scheduling in distributed systems. In *Proceeding of the IEE International Conference on Computational Intelligence, Modelling and Simulation*, pages 265–269, 2009.

[KLW11] Henning Kagermann, Wolf-Dieter Lukas, and Wolfgang Wahlster. Industrie 4.0: Mit dem internet der dinge auf dem weg zur 4. industriellen revolution. *VDI nachrichten*, 13(1):2–3, 2011.

[Kni02] John C. Knight. Safety critical systems: Challenges and directions. In *Proceedings of the International Conference on Software Engineering*, page 547–550, New York, NY, USA, 2002. Association for Computing Machinery.

[Kop11] Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media, 2011.

[KP17] Vasileios Karagiannis and Apostolos Papageorgiou. Network-integrated edge computing orchestrator for application placement. In *IEEE International Conference on Network and Service Management*, pages 1–5, 2017.

[Krs09] Miroslav Krstic. *Systems & Control: Foundations and Applications Delay Compensation for Nonlinear, Adaptive, and PDE Systems*. Birkhäuser Boston, 2009.

[KSLP19] Vasileios Karagiannis, Stefan Schulte, Joao Leitao, and Nuno Preguica. Enabling fog computing using self-organizing compute nodes. In *International Conference on Fog and Edge Computing*, pages 1–10, 2019.

[KW07] Robert Kaiser and Stephan Wagner. The pikeos concept: History and design. *SysGO AG White Paper. Available: http://www. sysgo. com*, 2007.

[KZ09] Andrew Kornecki and Janusz Zalewski. Certification of software for real-time safety-critical systems: state of the art. *Innovations in Systems and Software Engineering*, 5(2):149–161, 2009.

[LB03] Giuseppe Lipari and Enrico Bini. Resource partitioning among real-time applications. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 151–158, 2003.

[LB10] Giuseppe Lipari and Enrico Bini. A framework for hierarchical scheduling on multiprocessors: from application requirements to run-time allocation. In *IEEE Real-Time Systems Symposium*, pages 249–258, 2010.

[LC02] B. Lincoln and A. Cervin. Jitterbug: a tool for analysis of real-time control performance. *Proceedings of the IEEE Conference on Decision and Control*, 2(December):1319–1324, 2002.

[LCK16] Paulo Leitão, Armando Walter Colombo, and Stamatis Karnouskos. Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges. *Computers in Industry*, 81:11 – 25, 2016.

[LCSW14] J Lin, Albert MK Cheng, Douglas Steel, and Michael Yu-Chi Wu. Scheduling mixed-criticality real-time tasks with fault tolerance. In *Workshop on Mixed Criticality Systems*, 2014.

[Lee08] Edward A Lee. Cyber physical systems: Design challenges. In *Proceeding og the IEEE international symposium on object and component-oriented real-time distributed computing*, pages 363–369, 2008.

[LJYZ17] Jianhua Li, Jiong Jin, Dong Yuan, and Hongke Zhang. Virtual fog: A virtualization enabled fog computing framework for internet of things. *IEEE Internet of Things Journal*, 5(1):121–131, 2017.

[LRL09] Karthik Lakshmanan, Ragunathan Rajkumar, and John Lehoczky. Partitioned fixed-priority preemptive scheduling for multi-core processors. In *Proceeding of the IEEE Euromicro Conference on Real-Time Systems*, pages 239–248, 2009.

[LT01] Marin Litoiu and Roberto Tadei. Real-time task scheduling with fuzzy deadlines and processing times. *Fuzzy Sets and Systems*, 117(1):35–45, 2001.

[Lui01] Lui Sha. Using simplicity to control complexity. *IEEE Software*, 18(4):20–28, 2001.

[LYD$^+$17] Jian-Qiang Li, F Richard Yu, Genqiang Deng, Chengwen Luo, Zhong Ming, and Qiao Yan. Industrial Internet: A survey on the enabling technologies, applications, and challenges. *IEEE Communications Surveys & Tutorials*, 19(3):1504–1526, 2017.

[Mal09] Rajib Mall. *Real-time systems: theory and practice*. Pearson Education India, 2009.

[MAS$^+$18] Rouhollah Mahfouzi, Amir Aminifar, Soheil Samii, Ahmed Rezine, Petru Eles, and Zebo Peng. Stability-aware integrated routing and scheduling for control applications in Ethernet networks. In *Proceeding of the Design, Automation & Test in Europe Conference*, pages 682–687, 2018.

[MAS$^+$19] Rouhollah Mahfouzi, Amir Aminifar, Soheil Samii, Petru Eles, and Zebo Peng. Security-aware routing and scheduling for control applications on ethernet tsn networks. *ACM Trans. Des. Autom. Electron. Syst.*, 25(1), November 2019.

[MBD20] Ilir Murturi, Mohammadreza Barzegaran, and Schahram Dustdar. A decentralized approach for determining configurator placement in dynamic edge networks. In *Second International Conference on Cognitive Machine Intelligence*, pages 147–156. IEEE, 2020.

[McC96] Brian M McCay. Some thoughts on the quality of a computer-based system's architecture. In *Proceedings of the IEEE Symposium and*

Workshop on Engineering of Computer-Based Systems, pages 228–234, 1996.

[MDPC10] A. Masrur, S. Drossler, T. Pfeuffer, and S. Chakraborty. VM-Based Real-Time Services for Automotive Control Applications. In *Proceeding of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 218–223, Aug 2010.

[MEA^{+}10] Malcolm S Mollison, Jeremy P Erickson, James H Anderson, Sanjoy K Baruah, and John A Scoredos. Mixed-criticality real-time scheduling for multicore systems. In *Proceeding og the IEEE international conference on computer and information technology*, pages 1864–1871, 2010.

[MG^{+}11] Peter Mell, Tim Grance, et al. The NIST definition of cloud computing. 2011.

[MH18] Magdi S. Mahmoud and Mutaz M. Hamdan. Fundamental issues in networked control systems. *IEEE/CAA Journal of Automatica Sinica*, 5(5):902–922, 2018.

[MKB18] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. Fog computing: A taxonomy, survey and future directions. In *Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives*, pages 103–130. Springer, 2018.

[MLD09] Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm. *OPC unified architecture*. Springer Science & Business Media, 2009.

[MNH^{+}15] Saad Mustafa, Babar Nazir, Amir Hayat, Sajjad A Madani, et al. Resource management in cloud computing: Taxonomy, prospects, and challenges. *Computers & Electrical Engineering*, 47:186–203, 2015.

[MNY^{+}18] Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch H. Glitho, Monique J. Morrow, and Paul A. Polakos. A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges. *IEEE Communications Surveys and Tutorials*, 20(1):416–464, 2018.

[MPC19] Shane D. McLean, Paul Pop, and Silviu S. Craciunas. Mapping and scheduling of real-time tasks on multi-core autonomous driving platforms. Technical report, Technical University of Denmark, January 2019.

[MSMB11] Thomas Moser, Wikan Danar Sunindyo, Munir Merdan, and Stefan Biffl. Supporting runtime decision making in the production automation domain using design time engineering knowledge. *Ontology and Semantic Web for Manufacturing*, pages 9–22, 2011.

[MSZ11] Rupak Majumdar, Indranil Saha, and Majid Zamani. Performance-aware scheduler synthesis for control systems. In *Proceedings of the ACM international conference on Embedded software*, pages 299–308, 2011.

[MWTP$^+$13] Asma Mehiaoui, Ernest Wozniak, Sara Tucci-Piergiovanni, Chokri Mraidha, Marco Di Natale, Haibo Zeng, Jean-Philippe Babau, Laurent Lemarchand, and Sébastien Gerard. A two-step optimization technique for functions placement, partitioning, and priority assignment in distributed systems. In *Proceedings of the ACM SIGPLAN/SIGBED conference on Languages, compilers and tools for embedded systems*, pages 121–132, 2013.

[MYV$^+$04] Pau Martí, José Yépez, Manel Velasco, Ricard Villà, and Josep M. Fuertes. Managing quality-of-control in network-based control systems by controller and message scheduling co-design. *IEEE Transactions on Industrial Electronics*, 51(6):1159–1167, 2004.

[Neb21] Nebbiolo Technologies, Inc. Nebbiolo. `https://www.nebbiolo.tech/`, 2021 (accessed April 1, 2021).

[NSL17] Nicolas Navet and Françoise Simonot-Lion. *Automotive embedded systems handbook*. CRC press, 2017.

[OA09] Fatma A. Omara and Mona M. Arafa. *Genetic Algorithms for Task Scheduling Problem*, pages 479–507. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[Ope21a] OpenFog Consortium. OpenFog reference architecture for fog computing. `https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf`, 2017 (accessed May 1, 2021).

[Ope21b] OpenStack. Documentation on Nova Scheduler. `https://docs.openstack.org/developer/nova/filter_scheduler.html`, 2020 (accessed May 7, 2021).

[OY02] Katsuhiko Ogata and Yanjuan Yang. *Modern control engineering*, volume 4. Prentice-Hall, 2002.

[PBO19] Maryam Pahlevan, Balakrishna Balakrishna, and Roman Obermaisser. Simulation framework for clock synchronization in time sensitive networking. In *Proceeding of the IEEE International Symposium on Real-Time Distributed Computing*, pages 213–220, 2019.

[PEPP04] Paul Pop, Petru Eles, Zebo Peng, and Traian Pop. Scheduling and mapping in an incremental design methodology for distributed real-time embedded systems. *IEEE Transactions on Very Large Scale Integration Systems*, 12(8):793–811, 2004.

[PG74] Gerald J Popek and Robert P Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412–421, 1974.

[PGOP14] Peter Priller, Werner Gruber, Niklas Olberding, and Dietmar Peinsipp. Towards perfectly scalable real-time systems. In *Proceeding of the International Conference on Computer Safety, Reliability, and Security*, pages 212–223. Springer, 2014.

[Pic06] Bob Pickles. Avionics Full Duplex Switched Ethernet (AFDX). *SBS Technologies*, 15(2):62–65, 2006.

[PIEP09] P. Pop, V. Izosimov, P. Eles, and Z. Peng. Design optimization of time- and cost-constrained fault-tolerant embedded systems with checkpointing and replication. *IEEE Transactions on Very Large Scale Integration Systems*, 17(3):389–402, 2009.

[PML$^+$19] Carlo Puliafito, Enzo Mingozzi, Francesco Longo, Antonio Puliafito, and Omer Rana. Fog computing for the internet of things: A survey. *ACM Transactions on Internet Technology*, 19(2):1–41, 2019.

[PMN$^+$16] Q. Perret, P. Maurere, E. Noulard, C. Pagetti, P. Sainrat, and B. Triquet. Temporal isolation of hard real-time applications on many-core processors. In *Proceeding of the IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 1–11, 2016.

[PO18] M. Pahlevan and R. Obermaisser. Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks. In *Procceding of the International Conference on Emerging Technologies and Factory Automation*, volume 1, pages 337–344, 2018.

[PRCS16] Paul Pop, Michael Lander Raagaard, Silviu S Craciunas, and Wilfried Steiner. Design optimisation of cyber-physical distributed systems using IEEE Time-Sensitive Networks. *IET Cyber-Physical Systems: Theory & Applications*, 1(1):86–94, 2016.

[PRGS18] Paul Pop, Michael Lander Raagaard, Marina Gutierrez, and Wilfried Steiner. Enabling fog computing for industrial automation through time-sensitive networking (TSN). *IEEE Communications Standards Magazine*, pages 1–7, 2018.

[Pro21] Proton Technologies AG. General Data Protection Regulation. `https://gdpr.eu`, 2021 (accessed April 1, 2021).

[PSS19] T. Park, S. Samii, and K. G. Shin. Design optimization of frame preemption in real-time switched ethernet. In *Proceeding of the Design, Automation Test in Europe Conference*, pages 420–425, 2019.

[PTO19] Maryam Pahlevan, Nadra Tabassam, and Roman Obermaisser. Heuristic list scheduler for time triggered traffic in time sensitive networks. *SIGBED Rev.*, 16(1):15–20, February 2019.

[PYKL11] Kyung Joon Park, Man Ki Yoon, Kyungtae Kang, and Chang Gun Lee. Scheduling and control co-design under end-to-end response time constraints in cyber-physical systems. *Proceeding of the IEEE Conference on Computer Communications Workshops*, pages 762–767, 2011.

[PZB$^+$21] Paul Pop, Bahram Zarrin, Mohammadreza Barzegaran, Stefan Schulte, Sasikumar Punnekkat, Jan Ruh, and Wilfried Steiner. The FORA Fog Computing Platform for Industrial IoT. *Information Systems*, 98:101727, 2021.

[QBP21] Jia Qian, Mohammadreza Barzegaran, and Paul Pop. Decomposing deep training solutions on fog computing platforms. In *To be submitted to ACM/IEEE Symposium on Edge Computing*, 2021.

[QH20] Jia Qian and Lars Kai Hansen. What can we learn from gradients? *arXiv preprint arXiv:2010.15718*, 2020.

[Rau14] Marvin Rausand. Reliability of safety-critical systems. *Theory and Applications; John Wiley & Sons, Inc.: Hoboken, NJ, USA*, 2014.

[RHS97] Minsoo Ryu, Seongsoo Hong, and M. Saksena. Streamlining real-time controller design: From performance specifications to end-to-end timing constraints. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, pages 91–99, 1997.

[RL19] Richard Rennie and Jonathan Law. *A dictionary of physics*. Oxford University Press, 2019.

[RLL94] David Rowe, John Leaney, and David Lowe. Defining systems evolvability-a taxonomy of change. *Change*, 94:541–545, 1994.

[RPC20] Niklas Reusch, Paul Pop, and Silviu S. Craciunas. Safe and Secure Configuration Synthesis for TSN-based Distributed Cyber-Physical Systems using Constraint Programming. Technical report, Technical University of Denmark, November 2020.

[RRW$^+$03] John Regehr, Alastair Reid, Kirk Webb, Michael Parker, and Jay Lepreau. Evolving real-time systems using hierarchical scheduling and concurrency analysis. In *Proceeding of the IEEE Real-Time Systems Symposium*, pages 25–36, 2003.

[RS19] Jan Ruh and Wilfried Steiner. The need for deterministic virtualization in the industrial internet of things. In *Proceedings of the Workshop on Fog Computing and the IoT*, page 26–30, New York, NY, USA, 2019. Association for Computing Machinery.

[Rus00] John Rushby. Partitioning in avionics architectures: Requirements, mechanisms, and assurance. Technical report, SRI International MENLO Park CA Computer Science Lab, 2000.

[SCEP09] Soheil Samii, Anton Cervin, Petru Eles, and Zebo Peng. Integrated Scheduling and Synthesis of Control Applications on Distributed Embedded Systems. *In Proceeding of the IEEE Design, Automation & Test in Europe Conference & Exhibition*, pages 57–62, 2009.

[Sch17] Klaus Schwab. *The fourth industrial revolution.* Currency, 2017.

[Sch21] Klaus Schwab. The fourth industrial revolution: what it means, how to respond. `https://www.weforum.org/agenda/2016/01/the-fourth-industrial-revolution-what-it-means-and-how-to-respond/`, 2016 (accessed March 3, 2021).

[SCS18a] R. Serna Oliver, S. S. Craciunas, and W. Steiner. IEEE 802.1Qbv Gate Control List Synthesis Using Array Theory Encoding. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 13–24, 2018.

[SCS18b] W. Steiner, S. S. Craciunas, and R. Serna Oliver. Traffic planning for time-sensitive communication. *IEEE Communications Standards Magazine*, 2(2):42–47, 2018.

[SCT13] Andy Stanford-Clark and Hong Linh Truong. MQTT for sensor networks (MQTT-SN) protocol specification. *International Business Machines Corporation version*, 1:2, 2013.

[SDD12] F. Smarra, A. D'Innocenzo, and M. D. Di Benedetto. Optimal co-design of control, scheduling and routing in multi-hop control networks. In *Proceeding of the IEEE Conference on Decision and Control*, pages 1960–1965, 2012.

[SGAN$^+$16] Zakaria Sahraoui, Emmanuel Grolleau, Mohamed Ahmed-Nacer, Driss Mehdi, and Henri Bauer. Antinomy between schedulability and quality of control using a feedback scheduler,. In *Proceeding of the ACM International Conference on Real-Time Networks and Systems*, pages 171–179, 2016.

[SGJM$^+$20] Jorge Sanchez-Garrido, Antonio Jurado, Luis Medina, Rafael Rodriguez, Eduardo Ros, and Javier Diaz. Digital electrical substation communications based on deterministic Time-Sensitive Networking over Ethernet. *IEEE Access*, 8:93621–93634, 2020.

[SGM$^+$16] Zakaria Sahraoui, Emmanuel Grolleau, Driss Mehdi, Mohamed Ahmed-Nacer, and Abdenour Labed. Predictive-delay control based on real-time feedback scheduling. *Simulation Modelling Practice and Theory*, 66:16–35, 2016.

[SGMC12] Reinhard Schneider, Dip Goswami, Alejandro Masrur, and Samarjit Chakraborty. QoC-oriented efficient schedule synthesis for mixed-criticality cyber-physical systems. In *Proceeding of the IEEE Forum on Specification and Design Languages*, pages 60–67, 2012.

[SHB14] Zach Shelby, Klaus Hartke, and Carsten Bormann. The constrained application protocol (CoAP). 2014.

[Sie08] Siemens Simatic. Profinet system description–system manual. *Issue A5E00298288-04*, 6, 2008.

[Sin07] Oliver Sinnen. *Task scheduling for parallel systems*, volume 60. John Wiley & Sons, 2007.

[SKR$^+$18] Olena Skarlat, Vasileios Karagiannis, Thomas Rausch, Kevin Bachmann, and Stefan Schulte. A framework for optimization, service placement, and runtime operation in the fog. In *IEEE International Conference on Utility and Cloud Computing*, pages 164–173, 2018.

[SL03] Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *Proceeding of the IEEE Real-Time Systems Symposium*, pages 2–13, 2003.

[SL08] Insik Shin and Insup Lee. Compositional real-time scheduling framework with periodic model. *ACM Transactions on Embedded Computing Systems*, 7(3):1–39, 2008.

[SLS95] Jay K. Strosnider, John P. Lehoczky, and Lui Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, 44(1):73–91, 1995.

[SLSS96] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin. On task schedulability in real-time control systems. In *Proceeding of the IEEE Real-Time Systems Symposium*, pages 13–21, 1996.

[Smi03] Clifton L Smith. Understanding concepts in the defence in depth strategy. In *Proceeding of the IEEE International Carnahan Conference on Security Technology*, pages 8–16, 2003.

[Son09] Ye-Qiong Song. Networked control systems: From independent designs of the network qos and the control to the co-design. *IFAC Proceedings Volumes*, 42(3):155 – 162, 2009.

[SP16] Wilfried Steiner and Stefan Poledna. Fog computing as enabler for the Industrial Internet of Things. *e & i Elektrotechnik und Informationstechnik*, 133(7):310–314, 2016.

[Spe91] CAN Specification. Bosch. *Robert Bosch GmbH, Postfach*, 50, 1991.

[SR61] WC Schultz and VC Rideout. Control system performance measures: Past, present, and future. *IRE Transactions on Automatic Control*, (1):22–35, 1961.

[SR94] Kang G Shin and Parameswaran Ramanathan. Real-time computing: A new discipline of computer science and engineering. *Proceedings of the IEEE*, 82(1):6–24, 1994.

[SSL89] Brinkley Sprunt, Lui Sha, and John Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, 1(1):27–60, 1989.

[SSS17] Daniel Simon, Alexandre Seuret, and Olivier Sename. Real-time control systems: feedback, scheduling and robustness. *International Journal of Systems Science*, 48(11):2368–2378, 2017.

[Ste10] W. Steiner. An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks. In *IEEE Real-Time Systems Symposium*, pages 375–384, 2010.

[Sto96] Neil R Storey. *Safety Critical Computer Systems*. Addison-Wesley Longman Publishing Co., Inc., 1996.

[SVLN13] Kristian Sandström, Aneta Vulgarakis, Markus Lindgren, and Thomas Nolte. Virtualization technologies in embedded real-time systems. In *Proceeding of the IEEE Conference on Emerging Technologies & Factory Automation*, pages 1–8, 2013.

[SWP90] Behrooz Shirazi, Mingfang Wang, and Girish Pathak. Analysis and evaluation of heuristic methods for static task scheduling. *Journal of Parallel and Distributed Computing*, 10(3):222–232, 1990.

[SZ13] Hang Su and Dakai Zhu. An elastic mixed-criticality task model and its scheduling algorithm. In *Proceeding of the IEEE Design, Automation & Test in Europe Conference & Exhibition*, pages 147–152, 2013.

[T^+06] SEI AADL Team et al. An extensible open source AADL tool environment (OSATE). *Software Engineering Institute*, 2006.

[TDDFD20] Koen Tange, Michele De Donno, Xenofon Fafoutis, and Nicola Dragoni. A systematic survey of industrial internet of things security: Requirements and fog computing opportunities. *IEEE Communications Surveys & Tutorials*, 22(4):2489–2520, 2020.

[Tec21] TechNavio. Electric Drives Market by End-users, Product, Power Rating, and Geography - Global Forecast 2019-2023. `https://www.technavio.com/report/electric-drives-market-industry-analysis`, 2019 (accessed May 1, 2021).

[TG11] Yu Chu Tian and Li Gui. QoC elastic scheduling for real-time control systems. *Real-Time Systems*, 47(6):534–561, 2011.

[The21] The Linux Foundation®. Xen Project. `https://xenproject.org`, 2019 (accessed May 1, 2021).

[THM02] H. Topcuoglu, S. Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, 2002.

[TSP15a] Domiţian Tămaş-Selicean and Paul Pop. Design optimization of mixed-criticality real-time embedded systems. *ACM Transactions on Embedded Computing Systems*, 14(3):50–55, 2015.

[TSP15b] Domitian Tamas-Selicean and Paul Pop. Design optimization of mixed-criticality real-time systems. *ACM Transaction on Embedded Computing*, 14(3):50–78, May 2015.

[TTT21] TTTech Computertechnik AG. Nerve. `http://tttech.com/products/industrial/industrial-iot/nerve`, 2019 (accessed May 1, 2021).

[TWZ$^+$17] T. Terzimehic, M. Wenger, A. Zoitl, A. Bayha, K. Becker, T. Müller, and H. Schauerte. Towards an industry 4.0 compliant control software architecture using IEC 61499 OPC UA. In *Proceeding of the IEEE International Conference on Emerging Technologies and Factory Automation*, pages 1–4, 2017.

[Ull75] Jeffrey D. Ullman. NP-complete scheduling problems. *Journal of Computer and System sciences*, 10(3):384–393, 1975.

[Uni21] Universidad Politécnica de Valencia. XtratuM hypervisor. `https://xtratum.org`, 2019 (accessed May 1, 2021).

[Ves07] Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceeding of the IEEE International Real-Time Systems Symposium*, pages 239–243, 2007.

[VG01] Frank Vahid and Tony D Givargis. *Embedded system design: a unified hardware/software introduction*. John Wiley & Sons, 2001.

[WCP+05] Wei Zheng, J. Chong, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli. Extensible and scalable time triggered scheduling. In *International Conference on Application of Concurrency to System Design*, pages 132–141, 2005.

[WHL+19] Y. Wang, F. Huang, Y. Li, B. Pan, and Y. Wu. Hierarchical scheduling and real-time analysis for vehicular time-sensitive network. In *Proceeding of the International Symposium on Computational Intelligence and Design*, volume 2, pages 23–26, 2019.

[Wik21a] Wikipedia, the free encyclopedia. Scheduling (computing). `https://en.wikipedia.org/wiki/Scheduling_(computing)`, 2021 (accessed April 1, 2021).

[Wik21b] Wikipedia, the free encyclopedia. Virtualization. `https://en.wikipedia.org/wiki/Virtualization`, 2021 (accessed April 1, 2021).

[Wil94] Theodore J Williams. The Purdue enterprise reference architecture. *Computers in industry*, 24(2-3):141–158, 1994.

[WMZ+11] Ying Wang, Dianfu Ma, Yongwang Zhao, Lu Zou, and Xianqi Zhao. An AADL-based modeling method for ARINC653-based avionics software. In *Proceeding of the IEEE Annual Computer Software and Applications Conference*, pages 224–229, 2011.

[WS12] Zhi Wen Wang and Hong Tao Sun. Control and scheduling co-design of networked control system: Overview and directions. In *Proceedings of the IEEE International Conference on Machine Learning and Cybernetics*, volume 3, pages 816–824, 2012.

[WSJD15] L. Wisniewski, M. Schumacher, J. Jasperneite, and C. Diedrich. Increasing flexibility of time triggered ethernet based systems by optimal greedy scheduling approach. In *Proceeding of the IEEE Conference on Emerging Technologies Factory Automation*, pages 1–6, 2015.

[WZS+14] Yizhuo Wang, Yang Zhang, Yan Su, Xiaojun Wang, Xu Chen, Weixing Ji, and Feng Shi. An adaptive and hierarchical task scheduling scheme for multi-core clusters. *Parallel Computing*, 40(10):611 – 627, 2014.

[XABC17] Yang Xu, Karl-Erik Arzen, Enrico Bini, and Anton Cervin. LQG-based control and scheduling co-design. *IFAC 2017*, pages 6069–6074, 2017.

[XCÅ16] Yang Xu, Anton Cervin, and Karl Erik Årzén. Harmonic Scheduling and Control Co-design. In *Proceedings of the IEEE International*

Conference on Embedded and Real-Time Computing Systems and Applications, pages 182–187, 2016.

[XCÅ18] Yang Xu, Anton Cervin, and Karl Erik Årzén. Jitter-Robust LQG Control and Real-Time Scheduling Co-Design. In *Proceedings of the American Control Conference*, pages 3189–3196. IEEE, 2018.

[XX11] Yang Xu and Xiaoyao Xie. Modeling and analysis of security protocols using colored petri nets. *JCP*, 6(1):19–27, 2011.

[XXL$^+$14] S. Xi, M. Xu, C. Lu, L. T. X. Phan, C. Gill, O. Sokolsky, and I. Lee. Real-time multi-core virtual machine scheduling in Xen. In *Proceeding of the International Conference on Embedded Software*, pages 1–10, Oct. 2014.

[YHQL15] S. Yi, Z. Hao, Z. Qin, and Q. Li. Fog Computing: Platform and Applications. In *Proceeding of the IEEE Workshop on Hot Topics in Web Systems and Technologies*, pages 73–78, 2015.

[YLH$^+$18] Wei Yu, Fan Liang, Xiaofei He, William Grant Hatcher, Chao Lu, Jie Lin, and Xinyu Yang. A survey on the edge computing for the internet of things. *IEEE Access*, 6:6900–6919, 2018.

[YLL15] Shanhe Yi, Cheng Li, and Qun Li. A survey of fog computing: concepts, applications and issues. In *Proceedings of the workshop on mobile big data*, pages 37–42, 2015.

[YLL18] Luxiu Yin, Juan Luo, and Haibo Luo. Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing. *IEEE Transactions on Industrial Informatics*, 14(10):4712–4721, 2018.

[YXC02] Yumin Zhang, Xiaobo Hu, and D. Z. Chen. Task scheduling and voltage selection for energy minimization. In *Proceedings of the IEE Design Automation Conference*, pages 183–188, 2002.

[ZBP01] Wei Zhang, Michael S Branicky, and Stephen M Phillips. Stability of networked control systems. *IEEE Control Systems Magazine*, 21(1):84–99, 2001.

[ZDN17] Yun-Bo Zhao, Hui Dong, and Hongjie Ni. Scheduling and control co-design for control systems under computational constraints. *IFAC-PapersOnLine*, 50(1):5881–5886, 2017.

[ZHY16] Xian Ming Zhang, Qing Long Han, and Xinghuo Yu. Survey on Recent Advances in Networked Control Systems. *IEEE Transactions on Industrial Informatics*, 12(5):1740–1752, 2016.

[ZLZ15] Keliang Zhou, Taigang Liu, and Lifeng Zhou. Industry 4.0: Towards future industrial opportunities and challenges. In *Proceedings of the IEEE International conference on fuzzy systems and knowledge discovery*, pages 2147–2152, 2015.

[ZS00] Khawar M Zuberi and Kang G Shin. Design and implementation of efficient message scheduling for controller area network. *IEEE Transactions on Computers*, 49(2):182–188, 2000.

[ZSB$^+$12] Sergey Zhuravlev, Juan Carlos Saez, Sergey Blagodurov, Alexandra Fedorova, and Manuel Prieto. Survey of scheduling techniques for addressing shared resources in multicore processors. *ACM Computing Surveys*, 45(1):1–28, 2012.

[ZSEP19] Yuanbin Zhou, Soheil Samii, Petru Eles, and Zebo Peng. Scheduling optimization with partitioning for mixed-criticality systems. *Journal of Systems Architecture*, 98:191–200, 2019.

[ZSWY17] Dan Zhang, Peng Shi, Qing Guo Wang, and Li Yu. Analysis and synthesis of networked control systems: A survey of recent advances and challenges. *ISA Transactions*, 66:376–392, 2017.

[Zur05] Richard Zurawski. *Embedded systems handbook*. CRC press, 2005.

[ZYS$^+$09] Q. Zhu, Y. Yang, E. Scholte, M. D. Natale, and A. Sangiovanni-Vincentelli. Optimizing extensibility in hard real-time distributed systems. In *Real-Time and Embedded Technology and Applications Symposium*, pages 275–284, 2009.

[ZZZ$^+$13] Qi Zhu, Haibo Zeng, Wei Zheng, Marco DI Natale, and Alberto Sangiovanni-Vincentelli. Optimization of task allocation and priority assignment in hard real-time distributed systems. *ACM Transactions on Embedded Computing Systems*, 11(4):1–30, 2013.