TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Realzeit-Computersysteme

Synthesizing Communication-Centric Automotive Cyber-Physical Systems

Licong Zhang

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Die Dissertation wurde am 31.07.2017 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 23.05.2018 angenommen.

Abstract

Recent development in the automotive industry is currently revolutionizing the functionality of modern vehicles. Especially in the domain of infotainment and driver assistance systems, increasingly more new functions are being developed to make the cars safe, intelligent, convenient and comfortable for the drivers and passengers. With this enrichment of functionality, the E/E architecture of such system has also grown substantially in both size and complexity. This development introduces new challenges for meeting traditional requirements like real-time capability, resource efficiency due to the increasing size and complexity as well as emerging requirements like flexibility, adaptability, safety and security. To be able to address these challenges and scale into the future, the automotive E/E architecture also has to evolve. Towards this, several developing trends have been gathering increasing momentum, including, e.g., ECU consolidation, application of automotive Ethernet, adaptive platform, connectivity and Cyber-Physical System (CPS). Simultaneously, new design/synthesis approaches as well as frameworks need to be developed to facilitate the evolution of the E/E architecture along these directions and meet the existing and emerging design requirements.

This thesis addresses the problem of synthesizing communication-centric automotive CPS. The automotive E/E architecture is of a distributed nature where the Electronic Control Units (ECUs) are connected through different bus systems. The communication system plays an important role in the functional performance and non-functional properties of the applications and often becomes the bottleneck problem in the design of such systems. This thesis investigates the design of automotive CPS with the focus on the communication bus systems. In particular, it proposes three approaches addressing broadly three different requirements in the design of such systems, namely the real-time capability, flexibility/reconfigurability and resource efficiency.

The first approach addresses the problem of co-synthesizing task and communication schedules for an Ethernet-based time-triggered system. Ethernet has recently emerged as one promising candidate for the next generation automotive communication system. Towards the real-time requirements, some recent Ethernet protocols like the Time-Sensitive Networking (TSN) offer time-triggered communication, which is suitable for safety-critical and time-critical applications with stringent timing requirements. The proposed approach combines the task scheduling and communication scheduling in a synchronized manner and optimizes schedules according to application-level timing objectives. It further takes into account a number of Ethernet-specific timing parameters such as synchronization precision. The schedule synthesis problem is translated into a Mixed-Integer Programming (MIP) problem. This approach is able to handle one or multiple timing objectives such as application response time, end-to-end delay and their combinations and is scalable to systems of industrial size.

The second problem addressed in this thesis is the schedule management for the case of Plug-and-Play (PnP) and software update. In recent years, increasingly more new software applications are deployed in cars and this trend is expected to continue. However, the design and development cycle of software is much shorter than the life cycle of a vehicle and thus the functionality of a car could easily become outdated. Therefore, there have been emerging requirements on the software update and deployment of new software applications after sales. Towards this, the underlying E/E architecture has to offer a certain level of flexibility. One important issue here is the allocation of computation and communication resources. Addressing this problem, this thesis proposes a schedule management framework to obtain, synthesize and manage schedules efficiently online for Ethernet-based time-triggered systems in the automotive context. This framework is based on a client-server architecture and each side consists of a web module, a synthesis module and a configuration pool. It utilizes the Internet access of modern vehicles to exploit the computation and storage capacity on the server in a cloud-computing manner and can facilitate the reuse of generated schedule sets. In the synthesis module, a four-stage strategy is introduced to reduce the synthesis time and the disturbance to existing applications.

A further problem considered is the resource-efficient design in the context of CPS. Towards this, an approach to design resource-aware CPS over hybrid communication bus is proposed. Such a bus protocol offers both time-triggered (TT) and event-triggered (ET) communication. The TT communication offers higher timing predictability, which can potentially be translated into better control performance. However, such resource is often quite limited. Towards this, a resource-aware switching scheme for distributed embedded control applications is introduced in this thesis and illustrated using the FlexRay protocol. In the proposed approach, a combination of TT and ET communication is used to reject a disturbance and the strategy allows a control application to reside on the TT communication for an amount of samples that would be optimal for the overall control performance. Furthermore, it allows a control application to release TT resource before it is settled to make more efficient utilization of the resource. The proposed scheme involves both a control design method that optimizes the control performance and guarantees the switching stability and an online scheduling algorithm to dynamically determine the allocation of the TT communication to multiple control applications at run-time to optimize overall control performance. It further addresses the implementation challenges of the dynamic switching between TT and ET communication on the FlexRay protocol.

In summary, this thesis addresses the design/synthesis of the distributed automotive CPS. It proposes three new design approaches that address mainly the requirements on real-time capability, flexibility/reconfigurability and resource efficiency respectively. The approaches target at emerging fields like automotive Ethernet, Plug-and-Play and CPS and therefore quite relevant for the future. The methods proposed in this thesis might serve as the basis for further design solutions towards new in-vehicle communication architectures, adaptive platforms and cross-layer design in the CPS oriented fashion for the next generation E/E architecture that scales to the evolving requirements in the automotive domain.

Kurzfassung (German Abstract)

Die jüngste Entwicklung in der Automobilindustrie revolutioniert derzeit die Funktionalität moderner Fahrzeuge. Vor allem im Bereich der Infotainment- und Fahrerassistenzsysteme werden immer mehr neue Funktionen entwickelt, um die Autos sicher, intelligent und bequem für die Fahrer und die Passagiere zu machen. Mit dieser Anreicherung der Funktionalität ist die E/E-Architektur eines solchen Systems auch in Größe und Komplexität erheblich gewachsen. Diese Entwicklung stellt neue Herausforderungen für die Erfüllung traditioneller Anforderungen wie Echtzeitfähigkeit, Ressourceneffizienz durch die zunehmende Größe und Komplexität und auch entstehender Anforderungen wie Flexibilität, Adaptivität, Safety und Security vor. Um diese Herausforderungen zu begegnen und in die Zukunft zu skalieren, muss sich auch die E/E-Architektur weiterentwickeln. Auf diese Weise haben einige Entwicklungstrends an Schwung gewonnen, darunter z.B. ECU-Konsolidierung, Einsatz von Automotive Ethernet, adaptive Plattform, Konnektivität und Cyber-Physisches System (CPS). Gleichzeitig müssen neue Design-/Syntheseansätze sowie Frameworks entwickelt werden, um die Evolution der E/E-Architektur in diesen Richtungen zu erleichtern und die bestehenden und aufkommenden Designanforderungen zu erfüllen.

Die vorliegende Doktorarbeit befasst sich mit dem Problem der Synthese von kommunikationszentrierten Automotiven CPS. Die Automotive E/E-Architektur hat eine verteilte Struktur, wo die Steuergeräte (ECUs) über unterschiedliche Bussysteme verbunden sind. Das Kommunikationssystem spielt eine wichtige Rolle bei der funktionalen Leistung und den nicht funktionalen Eigenschaften der Applikationen und wird oft zum Engpassproblem bei der Entwurf solcher Systeme. Diese Doktorarbeit untersucht den Entwurf von Automotiven CPS mit dem Fokus auf die Kommunikationssysteme. Insbesondere sind drei Ansätze beim Entwurf solcher Systeme vorgestellt, die sich mit drei verschiedener Anforderungen beschäftigen, nämlich der Echtzeitfähigkeit, der Flexibilität/Rekonfigurierbarkeit und der Ressourceneffizienz.

Der erste Ansatz befasst sich mit dem Problem der Co-Synthese von Task- und Kommunikationsschedules für ein auf Ethernet-basiertes zeitgesteuertes System. Ethernet hat sich kürzlich als ein vielversprechender Kandidat für die nächste Generation Automotive Kommunikationssystem entstanden. Für die Echtzeitanforderungen bieten einige neuere Ethernet-Protokolle wie Time-Sensitive Networking (TSN) eine zeitgesteuerte Kommunikation an, das für sicherheitskritische und zeitkritische Anwendungen mit strengen Timing-Anforderungen geeignet ist. Der vorgestellte Ansatz kombiniert die Task-Scheduling und die Kommunikationsscheduling in einer synchronisierten Weise und optimiert die Schedules nach den Timing-Zielen auf der Anwendungsebene. Eine Reihe von Ethernet-spezifischen Timing-Parametern wie Synchronisationsgenauigkeit werden auch berücksichtigt. Das Problem der Schedule-Synthese wird in ein Mixed-Integer Programming (MIP) Problem übersetzt. Dieser Ansatz ist in der Lage, ein oder mehrere zeitliche Ziele wie Reaktionszeit der Anwendung, End-to-End-Latenz und ihre Kombinationen zu behandeln und ist skalierbar auf Systeme der industriellen Größe.

Das zweite Problem, das in dieser Doktorarbeit behandelt ist, ist das Schedule-Management für den Fall von Plug-and-Play (PnP) und Software-Update. In den vergangenen Jahren werden immer mehr neue Softwareanwendungen in Autos eingesetzt und dieser Trend wird voraussichtlich fortsetzen. Allerdings ist der Entwurfs- und Entwicklungszyklus von Software viel kürzer als der Lebenszyklus eines Fahrzeugs und somit könnte die Funktionalität eines Autos leicht veraltet sein. Daher gibt es neue Anforderungen an Software-Update und die Installation neuer Software-Applikationen nach dem Verkauf. Zu diesem Zweck muss die zugrunde liegende E/E-Architektur ein gewisses Maß an Flexibilität anbieten. Ein wichtiges Thema hierbei ist die Allokation von Berechnungs- und Kommunikationsressourcen. Angesichts dieses Problems stellt diese Arbeit ein Schedule-Management-Framework vor, um Schedules effizient online für Ethernet-basierte zeitgesteuerte Systeme im automobilen Kontext zu erhalten, zu synthetisieren und zu verwalten. Dieses Framework basiert auf einer Client-Server-Architektur und jede Seite besteht aus einem Webmodul, einem Synthesemodul und einem Konfigurationspool. Es nutzt den Internet-Zugang von modernen Fahrzeugen, um die Berechnungs- und Speicherkapazität auf dem Server in einer Cloud-Computing Weise auszunutzen und die Wiederverwendung von generierten Schedules zu erleichtern. Im Synthesemodul wird eine vierstufige Strategie eingeführt, um die Synthesezeit und die Störung auf bestehende Anwendungen zu reduzieren.

Ein weiteres Problem, das behandelt ist, ist der ressourcenbewusste Entwurf in Kontext von CPS. Ein Ansatz zum Entwurf von ressourcenbewussten CPS über Hybrid-Kommunikationsbus ist vorgestellt. Ein solches Bus-Protokoll bietet sowohl zeitgesteuerte (TT) als auch ereignisgesteuerte (ET) Kommunikation. Die TT-Kommunikation bietet eine höhere Timing-Vorhersagbarkeit, die potenziell in ein besseres Regelverhalten umgesetzt werden kann. Allerdings ist diese Ressource oft sehr begrenzt. Zu diesem Zweck wird in dieser Doktorarbeit ein ressourceneffiziente Switchingsschema für verteilte eingebettete Regelungsapplikationen eingeführt und mit dem FlexRay Protokoll verdeutlicht. Bei dem vorgestellten Ansatz wird eine Kombination aus TT- und ET-Kommunikation verwendet und die Strategie ermöglicht es, dass eine Regelungsanwendung auf der TT-Kommunikation für eine Menge von Samples liegt, die für das gesamte Regelungsverhalten optimal ist. Darüber hinaus ermöglicht er eine Regelungsanwendung, TT-Ressource freizugeben, bevor sie eingeschwungen wird, um eine effizientere Nutzung der Ressource zu ermöglichen. Das vorgestellte Schema beinhaltet sowohl eine Regelerentwurfsmethode, die das Regelsverhalten optimiert und die Switchingstabilität gewährleistet, und einen Online-Scheduling-Algorithmus, um die Allokation der TT-Kommunikation zu mehreren Regelungsapplikationen zur Laufzeit dynamisch zu bestimmen, um das gesamte Regelverhalten zu optimieren. Der Ansatz befasst sich darüber hinaus mit der Implementierungsherausforderung der dynamischen Umschaltung zwischen TT und ET Kommunikation auf dem FlexRay Protokoll.

Zusammenfassend behandelt diese Doktorarbeit die Entwurf/Synthese der verteilten Automotiven CPS. Die stellt drei neue Entwurfsansätze vor, die vor allem die Anforderungen an Echtzeitfähigkeit, Flexibilität/Rekonfigurierbarkeit und Ressourceneffizienz behandeln. Die Ansätze zielen auf entstehende Felder wie Automotive Ethernet, Plug-and-Play und CPS, und deswegen sind für die Zukunft relevant. Die in dieser Doktorarbeit vorgestellten Methoden könnten als Grundlage für weitere Lösungen für neue Kommunikationsarchitekturen, adaptive Plattformen und CPS-orientiertes Cross-Layer-Design für die E/E-Architektur der nächsten Generation dienen, die die wandelnde Anforderungen in dem Automobilbereich erfüllen wird.

Acknowledgements

This thesis is the result of the research work conducted at the Chair of Real-Time Computer Systems at the Technical University of Munich. It would not have been possible without the support of many people.

First of all, I would like to express my sincere gratitude to Prof. Samarjit Chakraborty for his guidance, support, advice and also encouragement. I would like to thank him for introducing me into this interesting topic and guiding me through the whole research work. I would also like to thank Prof. Petru Eles for agreeing to be the reviewer and co-examiner of this thesis and Prof. Andreas Jossen for heading the examination committee.

I would also like to thank all my colleges at the Chair of Real-Time Computer Systems. It has been a great pleasure to know them and work with them. Many research works have resulted from the discussions and collaborations with them. In addition, they have made the work at RCS much more enjoyable.

I am also grateful for the colleges that I have worked with from the RACE project and from other institutions that I have collaborated with. I have certainly benefited a lot from the collaborations on various challenging and interesting topics.

Finally, I would like to thank my family for the continuous support as well as encouragement.

Contents

Chapter 1

Introduction

Modern vehicles are becoming increasingly safer, more intelligent and driver friendly. In recent years, increasingly more new functions in domains like telematics/infotainment, Advanced Driver Assistance System (ADAS) are deployed in cars to assist the drivers. Examples of such functions include the navigation system, better Human Machine Interface (HMI), head-up display, surround view, parking assistant, Adaptive Cruise Control (ACC), Lane Departure Warning (LDW), etc. This trend of increase and enhancement of functionality is going to continue and autonomous and self-driving vehicles will soon become a reality. Underneath these functions that can be experienced by the user lies the Electrical/Electronic (E/E) architecture, which provides the basic services like the computation and communication for these functions. To keep up with the development pace on the functional level and be able to scale to the future, the underlying E/E architecture also has to evolve. Towards this, the requirements and challenges out of the recent development trends have to be addressed. These requirements include, e.g., real-time capability, resource-efficiency, flexibility, adaptivity, safety, security, etc. This thesis at hand proposes approaches to address several such design requirements and challenges on the topic of synthesis of communication-centric automotive Cyber-Physical System (CPS). In particular, these approaches synthesize schedules and partly also control parameters for distributed automotive E/E systems based on Ethernet and FlexRay towards real-time capability, reconfigurability and resource-efficiency. This chapter provides an introduction to the background, problem setting and the motivation of the thesis.

Chapter outline: This chapter is organized as follows. Firstly, the motivation of this thesis is explained in Section 1.1. This is followed by the introduction to the background in Section 1.2, including the automotive E/E architecture, the in-vehicle communication networks and the software architecture. Then a summary of the of the current and future development trends as well as the design requirements and challenges is provided in Section 1.3. Section 1.4 and Section 1.5 explain respectively two central topics related to this thesis, namely the scheduling and schedule synthesis problem and the design of CPS. Section 1.6 then summarizes the main contributions of this thesis. Finally, the organization of the thesis and the list of corresponding publications are provided in Section 1.7.

1.1 Motivation

The automotive E/E system is currently at the forefront of innovations from various disciplines like computer science, electrical engineering, control and mechanical engineering. Modern vehicles are being equipped with software-based applications of increasing sophistication, intelligence and connectivity, which considerably enhance the safety, convenience and comfort of the driving experience. Vehicle dynamics control systems, ADASs, navigation and connected telematics systems are just a few examples of such applications. On the functional level, more efficient algorithms in areas like computer vision, machine learning, Global Positioning System (GPS), augmented reality have contributed much to the rapid development in the domain of telematics, ADAS and autonomous driving. On the system level, the development and deployment of these software applications are made possible by more powerful hardware devices [1], better communication technologies and the advancement in software technologies.

The drastic growth of number and complexity of applications on the functional level has also profound influence on the underlying E/E architecture. Firstly, as a result of such development, the size and complexity of the automotive E/E architecture has been steadily increasing. Already in 2009, a premium-class vehicle could contain up to 100 Electronic Control Units (ECUs) running about 100 million lines of code [2]. As this trend of increasing size and complexity continues, and it will certainly do [3, 4], there will be many challenges that need to be addressed for the E/E architecture to be scalable for the future. Such challenges are on a wide spectrum of aspects, for example communication network, software architecture, Operating System (OS), hardware, etc. New or stronger requirements on issues like real-time capability, flexibility, scalability, resource-efficiency, safety and security need to be met.

On the other hand, the recent developments both inside and outside the automotive domain also offer opportunities and new technologies that can be leveraged on to address the challenges. For example, the development in the topic of connectivity has enabled modern cars the connection to other cars, intelligent infrastructure units and even backend servers. This connectivity can be utilized to get access to information and outsource computation or storage demands. A further one is the development in the communication technologies. The communication system in the automotive domain has been restricted to the domain-specific protocols like Controller Area Network (CAN), FlexRay and Media Oriented Systems Transport (MOST), which are tailored to the specific requirements in the automotive domain. However, increasingly more communication technologies from other domains can be applied to the automotive systems, for example Ethernet and higher layer protocols like Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Internet Protocol (IP) and Service Oriented Communications (SOC) can be either directly used in or adapted to the automotive domain. Thirdly, control applications take up a fair amount of applications in a car. Recent development in CPS-oriented design methods might also contribute to the cross-layer design in automotive E/E systems for more efficient design. Besides these, the safety topic can be approached by methods inspired from the avionics domain (like redundancy, Safety Integration Levels (SILs)), and to address the security topic, existing methods in network security can be exploited.

The automotive E/E architecture is a broad field and it is difficult to address all the aforementioned problems and aspects within the scope of one thesis. In the context of this thesis, the main focus is on the aspects of real-time capability, flexibility/reconfigurability and resource-efficiency. The real-time capability is one essential requirements in industrial embedded systems, because many of the applications here are safety- and time-critical. For these applications, not only the functional correctness, but also the non-functional correctness like timing need to be guaranteed. Flexibility/reconfigurability of the E/E system is a relatively new yet increasingly more important topic. New functionalities, particularly in the ADAS and infortainment domain, are developed with an accelerating pace. The ability of automotive E/E system to allow software updates and installation of new functions has therefore become an important future requirement. Furthermore, as the system grows in size and complexity and as increasingly more applications are integrated and consolidated on the processing units, resource-efficient design represents another essential design requirement. Therefore, this thesis presents three approaches related to the synthesis of schedules for the software tasks and the messages in the automotive E/E, with a particular focus on the communication network. These three approaches cover the three aforementioned fronts of design requirements and address the corresponding design and engineering challenges.

1.2 Background

1.2.1 Automotive E/E Architecture

The automotive E/E system consists of different electrical, electronic and software components. The architecture of such a system can be divided into several layers. For example, [3] describes four different layers of the E/E architecture, namely the function scope, the function and software architecture, the communication architecture, the component architecture. The functional scope [3] deals with the functional software, which can directly be experienced by the customers. Below the function scope is the function and software architecture [3], where the software functions are treated as blocks with input and output interfaces and several blocks may have data dependency between them. Here the actual function a software block is performing is abstracted and only the interfaces and the dependencies are important. On the communication architecture layer [3], the software blocks are then partitioned and mapped onto the ECUs. The data between the blocks are mapped onto the communication buses. The lowest layer is the topology component layer [3], where the ECUs are connected by bus systems and the power supply networks. In the scope of this thesis, we are primarily concerned with the function and software architecture layer and the communication layer.

The E/E architecture of a vehicle consists mainly of a number of ECUs, which are processors that the computation part of the E/E system is mapped on. These ECUs are connected through bus systems, where data between the ECUs are transmitted. The ECUs are typically organized in several different clusters (or functional domains [5, 6]), where each cluster is responsible for a specific category of applications. Examples of functional domains include the powertrain domain (e.g., engine and transmission control), the chassis domain (e.g. steering and braking control), the body and comfort domain (e.g., the control of doors, seats, lighting and air conditioning) and the infotainment domain (e.g., telematics and entertainment). In general, different functional domains have different requirements on the communication [6]. For example, in the chassis domain, the data for control applications are transmitted. These data are usually small (e.g., several bytes) but have stringent requirement on the timing properties (e.g., latency, jitter, etc.). On the other hand, the amount of data that needs to be transmitted on the bus systems in the infotainment domain is much larger, but the requirement on the timing properties is less strict. Therefore, the bus system used to connect different clusters are usually different.

A software *application* can consist of one or more software components that are executed on the ECUs. The whole application performs an independent function, e.g., a control loop. Each software component is a sub-module of an application and the components may have data dependencies. The application can be aggregated on one single ECU or distributed over multiple ECUs. The software components are implemented as *tasks*, where each task represents the execution of a piece of software code. If the component is executed more than once, each execution instance can be referred to as a *task instance*, denoting one occurrence of the task. A task can be periodic, i.e., triggered periodically by a schedule table, or event-triggered, i.e., it gets triggered only by the occurrence of specific events. The data between the tasks are transmitted over the communication system. Therefore the ECUs in each cluster are connected through one or more bus systems. The software on the ECUs consists of multiple layers. The software applications performing the actual functions, e.g., related to the control of the vehicle or interaction with the driver, are on the *application software layer*. Below the applications, there could also be one or

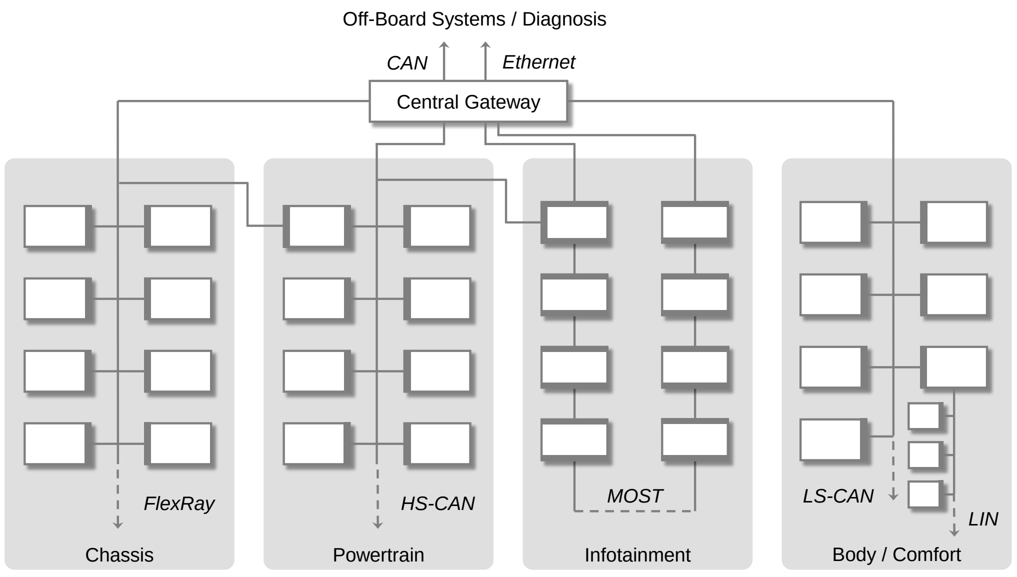

Figure 1.1: This figure shows a schematic example of an automotive E/E architecture in terms of communication bus systems. This figure is adapted from [7].

more layers of software components that provide the basic services to the application software, e.g., operating system, communication driver and Input/Output (I/O) drivers. The in-vehicle communication and the software architecture will be explained in detail in the following part of this chapter.

1.2.2 In-Vehicle Communication and Bus Systems

Typical bus protocols include CAN, Local Interconnect Network (LIN), MOST, FlexRay and Ethernet. Each bus protocol has its advantages and is suitable for functional domains with certain requirements. For example, the FlexRay or High-Speed-CAN is primarily used for the chassis domain, High-Speed CAN for the powertrain, Low-Speed CAN and LIN for the body domain and MOST and Ethernet for the infotaiment domain. The ECU/bus clusters are then connected to a central gateway, through which the data can be forwarded between different clusters, as shown in Figure 1.1. The in-vehicle communication forms one of the most important design aspects of the automotive E/E architecture.

FlexRay
FlexRay is a bus protocol developed by the FlexRay consortium and published in 2005 as version 2.1 [8] and in 2010 as version 3.0 [9]. It is hybrid protocol offering both time-triggered and event-triggered communication. It is organized as a series of communication cycles. In each cycle, a static segment implements the Time Division Multiple Access (TDMA) scheme, where messages are assigned pre-defined time slots. An optional dynamic segment employs the event-triggered Flexible Time Division Multiple Access (FTDMA)

scheme. Therefore, the FlexRay protocol combines both the advantages like determinism of the time-triggered communication and those like efficient resource utilization of the event-triggered communication. To support the time-triggered scheme, the clocks of the network nodes are synchronized to provide a global time. FlexRay allows the use of line topology as well as star topology and provides two channels (Channel A and B) that could be utilized either for bandwidth enhancement or redundancy [10, 11]. Due to the deterministic nature of the FlexRay (static segment), it is suitable for transmission of safety-critical data with strict timing requirements. On the other hand, due to the higher cost, compared to, e.g., CAN bus, its usage is still quite limited. However, FlexRay still gains ground in the chassis domain [10] in many types of cars produced by Original Equipment Manufacturers (OEMs) like BMW [7].

Ethernet
Ethernet is originally widely used for connecting computers and has a long history dating back to 1973 [12]. The Ethernet protocol has undergone a long process of development. Originally the Ethernet protocol features a Carrier Sense Multiple Access/Collision Detection (CSMA/CD) Media Access Control (MAC) with half-duplex links, where messages have to content for channel access and in the case of a collision, all messages will wait for a random backoff period [12]. Therefore, the message transmission of the CSMA/CD based Ethernet is not deterministic at all. Since then, full-duplex links are introduced to detach the sending and receiving link and switches (or bridges) are introduced to forward the Ethernet frames. In addition, priority operation is introduced to differentiate frames of different timing requirements [13]. Therefore, full-duplex switched Ethernet becomes a point-to-point network and has no collision between the network nodes anymore. The contention for communication bandwidth is transformed into the queueing time of messages in the output port of the switches.

Recent development of the IEEE protocol family include Audio Video Bridging (AVB), Time-Sensitive Networking (TSN) protocols, where both are a collection of amendments to the existing Ethernet protocols. The AVB, incorporated in the IEEE802.1Q standard [14], introduces several new features like clock synchronization, stream reservation protocol and the credit-based shaper. The AVB thus allows bandwidth reservation for the streams (e.g., of audio and video data) and avoids the starvation of low priority traffic by the streaming data using the credit-based fair queueing. The TSN [15] protocol is the successor of AVB and offers new features to improve the real-time capability of the network. Currently, the TSN protocols have not yet been fully specified, but some parts are published, including, amongst others, the *Enhancements for Scheduled Traffic* (IEEE802.1Qbv) [16] and the *Frame Preemption* (IEEE802.1Qbu) [17]. The IEEE802.1Qbv amendment introduces the mechanism supporting scheduled time-triggered transmission of messages. This is achieved by closing the gates of transmission control for the non-time-triggered queues when time-triggered traffic arrives and opening the gates again once the transmission is finished [16]. The frame preemption allows more critical messages to preempt less critical messages so that the *blocking time* is reduced [17]. Both mechanisms are targeted at improving the real-time properties of time-critical messages and can be used either individually or in combination [17].

Besides the IEEE protocol family, there are also several proprietary Ethernet protocols. The most notable of them are PROFINET [18] and EtherCAT [19] in the industrial automation domain and the Avionics Full Duplex Switched Ethernet (AFDX) [20] and AS6802 (TTEthernet) [21] primarily targeted at the avionics domain. These protocols offer tailored solutions to the individual domain or application system. In general, the PROFINET and AS6802 protocols show some similarities to the TSN protocol, where the network traffic is divided into several categories and for the applications with most stringent real-time requirements, the time-triggered (or scheduled) traffic is used.

Currently, Ethernet is already used for diagnosis and applied in the infotainment domain, having the potential to replace MOST [11, 22]. In the future, it is considered a promising candidate for serving as the backbone between domains [23], and even be applied in the safety-critical domains. There are also already research platforms employing Ethernet as the dominant communication system [24]. However, there are still several hurdles to overcome for a wide application of Ethernet in the automotive domain. Firstly, the new Ethernet protocols like AVB and TSN are still relatively new. The suitability of these protocols are still yet to be proved, especially for the safety-critical domains, where real-time capability is usually an important requirement. Furthermore, compared to conventional bus systems, there are still not sufficient work on the design and analysis methods as well as tool support. Finally, cost reason also plays an important role, due to the cost-sensitive nature of the automotive industry. Implementing Ethernet means also making the corresponding hardware and software compatible. This introduces considerable cost on hardware/software components as well as the cost for development, testing and validation. The cost issue is partly addressed by the introduction of BroadR-Reach technology [25], featuring a Physical (PHY) layer based on an unshielded single twisted pair of wire, thus considerably reducing the wiring cost. Currently 100 Mbit/s is already available and deployed in some vehicles with an upgrade to 1 Gbit/s in the near future [26].

CAN, LIN and MOST

CAN bus was introduced by Robert Bosch GmbH in 1986 and has since then become one of the most widely used bus systems in automobiles. CAN offers different versions of data rates up to 1 Mbit/s [26]. Most used versions of CAN are the *High-Speed-CAN* [27] with a common data rate of 500 kbit/s [11] and the *Low-Speed-CAN* [28] with a common data rate of 125 kbit/s [11], although a range of different data rates are supported by the both. A payload size of up to 8 bytes per message is supported by CAN. The actual use of CAN in different domains depends strongly on the manufacturer and the vehicle type. In general, this bus finds prevalent deployment in the powertrain domain (High-Speed-CAN), the body domain (Low-Speed-CAN) and sometimes also in the chassis domain (High-Speed-CAN). CAN implements an event-triggered communication scheme and is a decentralized broadcast bus. The collision between different senders are resolved through the collision resolution mechanism known as Carrier Sense Multiple Access/Collision Resolution (CS-MA/CR). Pre-defined priorities are assigned to CAN messages and when two messages are sent simultaneously, the one with higher priority will gain access of the bus. Although CAN has been the most prevalent bus system used in the automotive domain, it has several limitations, which constrain its potential for recent and future applications like ADAS. One limitation is the limited bandwidth and payload size. Towards this, the extension of CAN with Flexible Date-Rate (CAN FD) [29] has been developed that increases the payload size to 64 bytes without increasing the transmission time. This is achieved by using a new frame format and switching the data rate inside a frame. This protocol is also backward compatible with CAN [26]. The other limitation is the non-determinism and possible long latencies and large jitter, especially for lower priority messages on a loaded bus, although timing analysis techniques for CAN are commonly known [30]. There have also been efforts addressing this issue. A notable extension is the Time Triggered CAN (TTCAN) [31] protocol. The TTCAN employs both time-triggered communication based on TDMA and event-triggered communication based on CSMA/CR. The time is organized in cycles and in each cycle, some time slots are reserved for certain ECUs for time-triggered transmission and some are offered for contention-based transmission. For the TDMA scheme, the network nodes are synchronized. However, due to no enhancement of the data rate and other reasons, TTCAN is not widely implemented [26]. It is expected that CAN will be replaced by FlexRay in the domain of chassis and powertrain [11].

The development of LIN started in the later 1990s by the LIN consortium to find a cheaper alternative to the Low-Speed-CAN bus for the body domain to connect simple nodes like doors, seats, etc. [11] The current version of the specification is Version 2.1 released in 2006. The LIN bus offers a bandwidth of up to 20 $kBits/s$ and implements a master/slave based communication [3, 11]. One LIN cluster consists of one master node and multiple slave nodes. The communication is primarily based on a polling mechanism where the master node sends periodically a header with message ID and the corresponding slave node sends the data in the response field. The header and the response together form a LIN frame and the time slots are configured to accommodate the frames. Such a frame is called an *unconditional frame*. The communication relation is pre-configured and defined in the LIN Description File (LDF). Since only one slave node is responding to a header simultaneously, there is not collision in this case. To support also event-triggered communication, the latest version of LIN has also added other frame types like the *event-triggered frame*, *sporadic frame* and *diagnostic frame* [11]. The event-triggered frame allows multiple slave nodes to respond simultaneously and if a collision is detected, the master node will initialize a round of polling with unconditional frame through all the slave nodes.

MOST is another commonly used bus protocol, primarily targeted at telematics and multimedia applications [11]. The most commonly used topology in MOST is a ring topology where the ECUs are connected point-to-point with a direction [11, 32]. One ECU serves as the *Master* and sends frames. The rest (*slaves*) synchronize to the master and read or write data into the frames [11]. The supported data rates of MOST are 25 Mbit/s (*MOST25*), 50 Mbit/s (*MOST50*) and 150 Mbit/s (*MOST150*). The protocol supports both time-triggered (synchronous) and event-triggered (asynchronous) communication. MOST is expected to be replaced by Ethernet in the infotainment domain [11, 22].

SOC and Middleware Solutions
Traditionally the configuration of the communication network in vehicles is quite static. Communication signals are mapped into Protocol Data Units (PDUs) and then into frames. The communication relation (e.g., the sender and receivers) and the schedules (or priorities) are pre-defined at design and development phase and usually cannot be changed afterwards. Thı́s paradigm might not be suitable for future applications and E/E architecture, which demand a more flexible communication paradigm. Therefore, there have been some efforts to apply and adapt SOC and middleware solutions for the in-vehicle communication. Examples of such efforts include the Scalable service-oriented Middleware over IP (SOME/IP) [33] and Data Distribution Service (DDS) [34]. Such protocols are based on the publish/subscribe mechanism and a network node can offer service of certain data (or *topic*) and others can subscribe and de-subscribe this data. Many of such protocols are based on Ethernet. But there are also solutions that are based on existing automotive bus protocols like CAN [35].

1.2.3 Software Architecture

A modern ECU is becoming increasingly complex in terms of software. For example, an ECU can contain multiple software components of different applications. In addition, an ECU needs to have I/Os and connection to the bus system. Therefore, besides the functional software components, there is a need for basis software components like OS and drivers and services for the I/Os and the bus protocols. To handle this complexity, the software of an ECU is usually divided into different layers. For example, there is usually a basis software layer between the hardware, i.e., the microcontroller, and the application software that implements the basic software components for the scheduling, I/Os and communication services, so that the microcontroller and the basic services are abstracted from the application software. Optionally there could also be an Runtime Environment (RTE) layer to handle the coordination and communication between the software components. Another important issue that needs to be addressed is the reusability of the the software. To address these issues, the automotive industry has spent much effort in standardizing the software architecture.

OSEK/VDX
In the 1990s, some Germany automotive OEMs and suppliers have formed the Offene Systeme und deren Schnittstellen fur die Elektronik in Kraftfahrzeugen (OSEK) consortium towards standardizing the software architecture [3]. Later this was merged with the french effort Vehicle Distribution eXecutive (VDX) and resulted in the OSEK/VDX standards [37]. The OSEK/VDX specifies several aspects of the software architecture, including the OS (OSEK-OS and OSEK-Time), the communication module (OSEK-COM), the

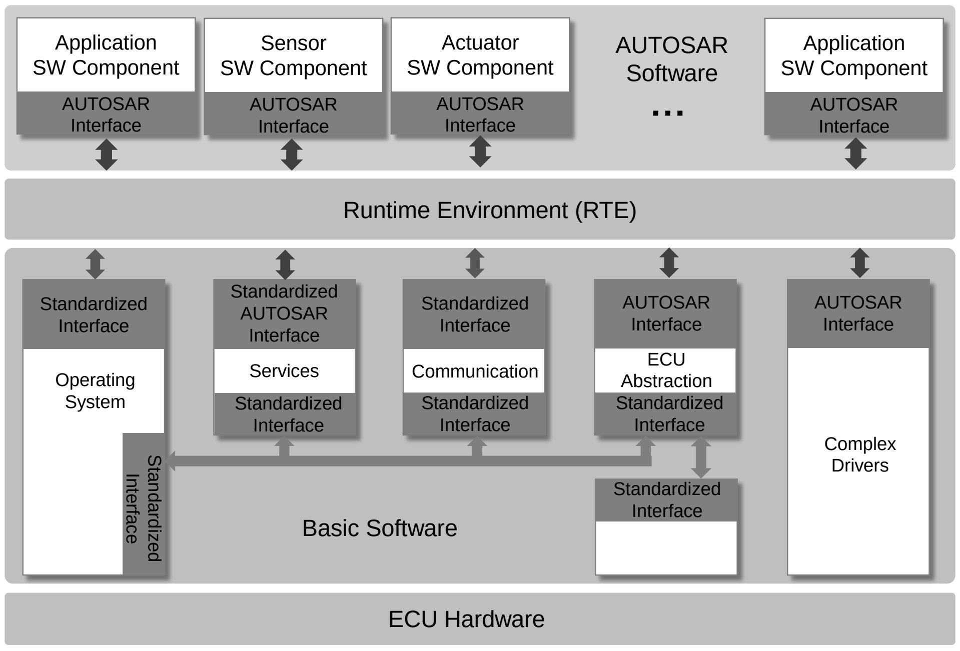

Figure 1.2: This figure shows the reference architecture from Automotive Open System Architecture (AUTOSAR). This architecture consists of three software layers, namely the basic software layer, the runtime environment and the application software layer. The interfaces between the software components are standardized. This figure is reproduced from [36].

network management (OSEK-NM) and the implementation language (OSEK-OIL). The OSEK-OS implements the event-triggered scheduling scheme [3, 38]. The OSEK-TIME is the time-triggered version of OSEK-OS [3].

AUTOSAR

The AUTOSAR standards were published by a world wide partnership [39]. Amongst other aspects, the AUTOSAR standards define a reference software architecture, the interfaces between the different components and development methods [39]. In terms of OS, the AUTOSAR-OS is backward compatible with the OSEK-OS. The reference architecture defined in the AUTOSAR standards is shown in Figure 1.2, where the software of an ECU is divided into three layers, namely the *basic software layer*, the *RTE* and the *application software layer*. The basic software layer provides the basic services and abstracts these services from the application software components. This layer can be further divided into the *Microcontroller Abstraction Layer*, the *ECU Abstraction Layer* and the *Service Layer*. The RTE serves as the coordination of the communication between the application software components and between the application software and basic software. The application software layer contains mainly the software components that perform actual functions, e.g., engine control software module. The communication interfaces between the software components are also defined by AUTOSAR. Here the interfaces are defined as *ports* and there are different types of ports, depending on the communication relationship, e.g., *sender-receiver* or *client-server.* The communication and data exchange between the components are defined using the *Virtual Functional Bus (VFB).* The VFB is a concept abstracted from the actual implementation of the communication, e.g., through different bus systems or inside the RTE, which depends on the actual mapping of the software components on the ECUs and the available communication bus systems. Through the standardization of the architecture and the component interfaces, modularity of the software components is achieved and as a result, it allows the reuse of software components and reduction of the complexity. For example, the basic as well as the application software components can be developed by different suppliers and integrated by the OEM. Furthermore, for different vehicle variants, where the underlying ECUs and buses are different, the same application software components can be reused.

1.3 Trends, Requirements and Challenges

1.3.1 Current and Future Trends

The automotive systems are currently undergoing rapid development and changes. These new developments are both on the application level and the system level. On the application level, increasingly more software-based systems are replacing those that are traditionally implemented with mechanical and hydraulic systems. Furthermore, new areas like the ADAS and autonomous driving, Car2X technology and connectivity as well as E-mobility are drawing increasingly more attention and are at the forefront of the innovations in the automotive domain. On the system level, topics like ECU consolidation, the introduction of Ethernet, Service Oriented Architecture (SOA) as well as safety and security are moving to the center of the stage. Here a few of the most important development trends, both on the application and system level that are related to the context of this thesis are listed and explained below. Certain topics also of significance, e.g., functional safety, security and E-mobility, are not explained here since they are not within the scope of this thesis.

Increasing Complexity and ECU Consolidation
One of the most obvious trend in the automotive embedded systems is the increase of the size and complexity of the E/E system, as shown in Figure 1.3 ,which is the result of increasingly more software applications deployed in vehicles. This trend is reflected in the steadily increasing number of ECUs, bus systems, signals and lines of software code in modern vehicles. The main challenge here is that this increase is not sustainable with current *federated* [40] architecture, where each ECU performs only specific functions. This is due to the reason of complexity management, limited resources (e.g., communication resource), cost and ultimately the weight and space in the vehicle. One possible way to address this

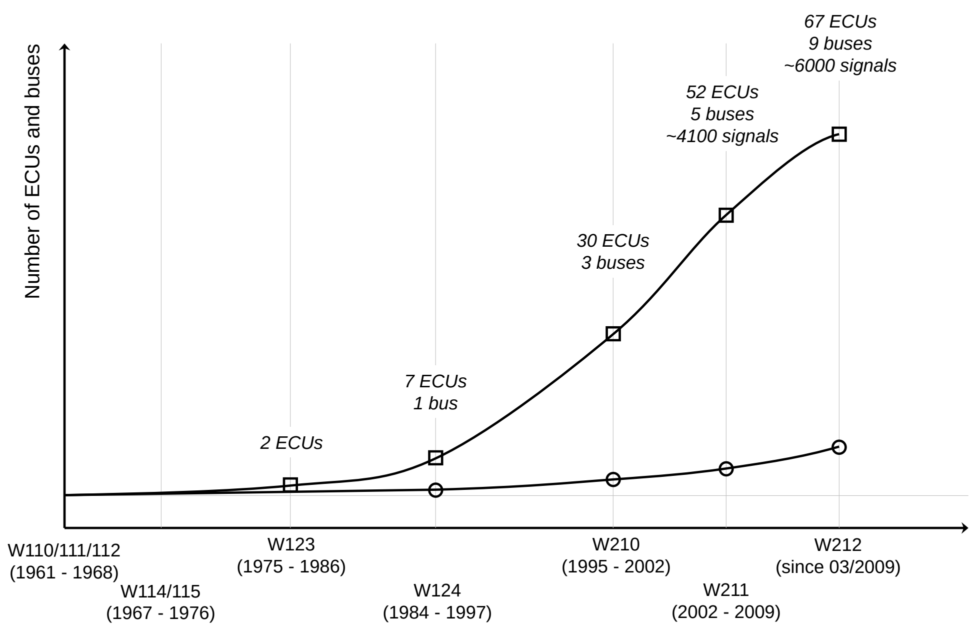

Figure 1.3: This figure shows the evolution of the E/E architecture of the Mercedes-Benz E-class in terms of number of ECUs, buses and signals. The data are provided in [3, 26].

problem is *ECU consolidation* [4, 41]. As pointed out in [40], in the future, the automotive E/E architecture is moving from the federated architecture towards an *integrated* architecture, where the ECUs are treated as computing platforms and multiple functions can be integrated onto one single ECU. With the ECU consolidation, the increase in the number of ECUs could be curbed and the centralization of the software functions also helps managing the complexity. Towards this, more sophisticated software architecture and middleware solutions are required to integrate the software applications. One example of such architecture is the system developed in the RACE project [24, 42]. The AUTOSAR standards also contribute to this topic by standardizing the interfaces of software components and specifying the reference software architecture.

ADAS and Autonomous Driving

ADAS and autonomous driving is one topic that certainly stands at the center of the stage in the automotive industry in the recent years. The development of ADAS roots back to 1978, where the first Anti-lock Braking System (ABS) system was deployed. Since then, increasingly more functions like Traction Control System (TCS), Electronic Stability Program (ESP) have been developed to help the drivers maintain and control the dynamics of the vehicle. These systems are also referred to as active safety systems. It is only recently, that more driver assistance systems like ACC, assistent/autonomous parking, traffic sign recognition, LDW, etc., are developed and deployed in the vehicles. These driver assistance systems are based primarily on cameras (mono-camera or stereo-camera), radars and Light Detection and Ranging (LIDAR) sensors. Recently, companies like Google [43] and Tesla also offer commercial autonomous driving vehicles [44, 45]. Besides the large impact of the ADAS and the autonomous driving systems on the safety and comfort of the driving experience, they also have profound impact on the underlying E/E architectures. Firstly, the rapid development in this domain means increasingly more software components, hardware components and data are being added to the E/E systems, contributing considerably to the increase in size and complexity. Secondly, the ADAS functions are usually computation- and data-intensive functions which requires much longer execution time and much larger data amount that needs to be processed and possibly transferred. Although, currently sensor data like camera images are not directly sent on the bus systems, such possibility still exists and need to be considered in the design trade-offs. Thirdly, though not all ADASs are safety-critical, many of them are. This means that the requirements of these applications on aspects like real-time capability, safety, reliability and security need to be met. Finally, current automotive E/E architectures are grouped into domains, and all the domains are connected to a central gateway, where the inter-domain communication is limited. However, many of the ADAS functions are cross-domain in nature [46]. These would introduce new challenges to the current domain-based architecture.

Plug-and-Play (PnP) and Software Update

Increasingly more new and advanced functions are being developed and deployed in vehicles, especially in the ADAS and infotainment domain. However, the design and development cycle of software and electronics is much shorter than the life cycle of a car, which can be up to about 15 years [1], not to mention the additional years of the design and development of the car. Therefore, with the current static architecture, it is difficult for a vehicle to be equipped with the newest software functions. It would be advantageous for the customer, if the new software functions can be installed or existing software functions be updated after the car has been sold. This topic is commonly referred to as software update and PnP. Currently, software updates are already available for some vehicles. However, this update is currently only limited to some specific ECUs, e.g., the Headunit. The software update and PnP of new applications are currently not yet standard practice. To enable this, there are several challenges that need to be addressed, including the connectivity for software download and dynamic platform that supports the deployment of the software and the reconfiguration of the system, e.g., the re-allocation of the computation and communication resources.

Connectivity

Another important trend in modern automotive industry is connectivity. Modern car now connect to other cars, road infrastructure units through wireless technologies [47]. This family of technologies are often referred to as Car2X technologies or Intelligent Transportation System (ITS). The connectivity of course offers a lot of advantages and at the same time brings new design challenges to the E/E systems. On one hand, the connectivity enables cars to connect to other vehicles and road side units in the surroundings for driver assistance or vehicle coordination functions. It also allows the vehicles the connection to the cloud and back-end servers, which can provide live information, online software updates and even computation capacity to the vehicles. On the other hand, it also poses challenges in terms of security, reliable communication and extra latencies if some functions are cloud-based.

Automotive Ethernet

Conventional bus systems in the automotive domain include CAN, LIN, MOST and FlexRay. These protocols are very different in nature and are designed to accommodate requirements of applications in different domains. Although these bus protocols were tailored to fulfill the requirements of their corresponding domain, as the size and complexity of the system increase, these bus protocols are slowly reaching their limits, and not meeting the future requirements in terms of higher bandwidth, flexibility, scalability, etc. Furthermore, although updating the current bus systems is possible (e.g., CAN FD and TTCAN), they do not provide a solution that can be scaled into the future. Therefore, the automotive industry has been exploring a new bus protocol that are sufficiently tested and proved and meet the steadily more complex requirements in the automotive domain. One promising candidate is the Ethernet protocol, which has been widely used in the Internet and office networks. Recent development in the Ethernet protocols like AVB [14] and TSN [15] can possibly also meet the requirements of the applications in the automotive domain. Another major advantage of Ethernet is that it is used and supported by a much larger community than the automotive industry. This means that the methods, tools, experiences and expertise of the other communities can be transferred to the automotive domain. One example is that the BroadR-Reach technology [25], which is de-factor standard PHY layer technology in automotive Ethernet, was developed by Broadcom. Although there has been a lot of work going on in the automotive Ethernet in recent years, there is, however, not yet a clear and standardized definition of what is automotive Ethernet. Instead, there are a collection of many relevant IEEE standards [14, 16, 17] that offer a wide spectrum of features and not all of the features are implemented in vehicles and which features to implement might depend on the OEMs. In addition, although there has been predictions that Ethernet is going to replace current bus systems, it is still not clear about the time line and art of this. It is, however, expected [11, 22] that Etherent will firstly replace MOST in the infotainment domain, then serve as a back-bone between the different domains [23]. Since the recent standards of Ethernet have only been finalized recently (AVB in 2011 and TSN only partially finalized in 2015 and 2016), there still needs to be considerable effort in the design, analysis methods and tool support for these new protocols before Ethernet can be deployed in full fledge in vehicles.

CPS

CPSs has emerged as one important research domain in the past decade [48–51]. The motivation behind this domain is that currently increasingly more control-centric embedded systems are deployed in real-life applications. As conventional embedded systems focus more on the side of the embedded computing units, the CPS paradigm emphasizes on the tight interaction between the computational units and the physical processes to be controlled. This paradigm enables the consideration of control design and embedded platform design in a holistic manner, thus preventing false assumptions of the other side and reducing the design conservativeness. By employing the CPS-oriented methods, it is usually possible to optimize the control and system design jointly according to certain objectives like control performance, resource utilization, etc [52, 53].

1.3.2 Design Requirements and Challenges

The developing trends in the automotive domain mentioned above bring new opportunities to explore and at the same time introduce new challenges to be addressed in the E/E systems. Important requirements as well as challenges concerning this thesis are listed as the following.

Real-Time Requirements
Real-time requirements refer mainly to the requirements on the timing properties of applications. These properties include for example the response time of the software tasks, the transport latency of the messages, the end-to-end latency and response time of the distributed applications, the jitter and etc. For applications with real-time requirements, some or all of these timing properties need to fulfill certain requirements, e.g., the values are smaller than a given threshold value, which is also known as the deadline.

Increasingly more applications in the automotive domain are control applications. Many of such applications are safety- and time-critical and therefore the real-time capability that the underlying E/E system can provide is one of the major requirements that need to be fulfilled. For example, for a given feedback control loop, the sampling period [54, 55] and the end-to-end delay [56–58] plays an important role in the performance of these applications. The airbag system is another example of application with stringent timing requirements, where the airbag needs to be activated fast enough after the crash to protect the driver and the passengers and the failure to comply with this requirement will lead to catastrophic results. A further reason for the real-time capability is the possibility of the optimization of the functional performance. The functional design of the applications sometimes relies on the timing properties that can be guaranteed by the software implementation of the applications [55, 56]. If better real-time properties can be provided, it is possible to design functions that can achieve better performance.

The real-time capability has always been one of the major requirements in the automotive embedded systems. However, as increasingly more safety-critical control applications are implemented in the vehicles and many of such applications are time-critical, further challenges are imposed. Firstly, increasingly more software and messages bring growing complexity for the design methods [26, 59]. Secondly, there are increasingly more applications that are of inter-domain nature [23, 46], e.g., some ADAS systems. This requires the cross-domain communication, which spans over two or more (heterogeneous) bus clusters thus complicates the design and analysis since possibly different bus protocols, gateways and processor scheduling schemes need to be considered.

Design and Analysis for Communication Protocols

The design and analysis for the communication protocols is an essential topic for the automotive E/E architecture. There could be several aspects of requirements on the communication bus systems. One of the most important is the timing properties. These include, e.g., the transport latency and jitter. Other important requirements include the bandwidth, fault-tolerance, cost, etc [11]. The communication protocols like CAN and FlexRay specify the basic mechanisms. However, there are still a number of parameters to be configured, e.g., priority, schedule, signal-to-frame mapping, etc. The analysis refers to the process of determining whether the requirements are met given a set of values for the parameters. For example, one important analysis here is the timing analysis. For safety- and time-critical systems, the worst-time timing guarantee needs to be provided to ensure the non-functional safety of the system. There have been numerous works on the timing analysis of bus protocols, including CAN [30], FlexRay [60, 61] and Ethernet [62–64], and end-to-end timing analysis methods [65–67]. The design of the communication refers to the process of choosing the values of the parameters so that the requirements are met, and possibly some objectives can be optimized. Towards this, efficient design methods are important. Manual or ad hoc design of the parameters can be a tedious and error-prone process. Automated design approaches are more efficient and safe in comparison. One important set of design methods is the *synthesis* methods. These methods usually model the communication protocols and, based on the models, use different synthesis or optimization methods to obtain the desired parameters [55, 59, 68, 69]. The actual method used depends a lot on the nature of the protocol. Section 1.4 will provide a detailed description of the synthesis methods.
There are several challenges for this aspect. Firstly, new protocols introduce new models, which need to be derived and efficiently formulated. For example, the modeling of the switched Ethernet is different from conventional broadcast bus systems. AVB and TSN standards introduce further features, which are quite challenging to model, both for the analysis and for the design problem. Secondly, the evolving requirements and increasing size also pose additional challenges. It is known that for both design and analysis methods, the computational effort required increases as the size of the system increases. Therefore, as the size of the system increases, the problem could become intractable using the existing methods. Timing analysis techniques usually involves a certain level of pessimism and this pessimism could also grow with the size of the system. More complex requirement, e.g., in the ADAS domain, also poses challenges. Furthermore, in future applications where the connection with the cloud is involved, the parameters and timing between the vehicle and the cloud need to be considered. In addition, incremental design is yet another challenge. The design of automotive systems usually follows an iterative design paradigm [70, 71]. This means that when new components are added to the system, the design of the existing components are kept as less changed as possible, to reduce the cost of re-design, re-test and re-validation. Such a scheme requires also extensible design [70, 72] as well as efficient incremental design methods [71].

Resource-Efficient Design
There are several types of resources in embedded systems, including, e.g., the computation resources, the memory resources, the communication resources and battery resources [52, 73]. The computation resource refers to the computation capacity of the processors. In the case of a single-core processor, the computation resource can be translated into the computation time. The execution of each software task requires a certain amount of time, and in the case of multiple tasks running on the same processor, a specific computation time needs to be allocated to each. There are typically two levels of memory on a processor, the cache, or on-chip memory, and the main memory. The cache is faster but of higher cost, and therefore can only be equipped with limited size [74]. The communication resource refers to the resource that is used to send messages between the processors and can generally be interpreted as the bandwidth of a communication bus or a network link. It should be pointed out that the exact definition of the computation, memory and communication resource depends strongly on the underlying mechanism or protocol (e.g., OS for the computation resource and bus protocol for the communication resource).

Resource limitation is one common problem in embedded systems. This is especially true for the cost-sensitive automotive industry. Aligned with the trend of ECU consolidation, there are increasingly more software components sharing the same processor or the same bus system. Although the processors are also becoming more powerful, computation resources are still limited. The communication resource on the other hand, can become a bottleneck problem even more easily. The processors can be replaced by more powerful ones and even additional processors can be added. The bus protocol specification usually stays static and adding an extra bus requires considerable more effort (e.g., in design, development, testing and validation). Therefore, the communication resources easily becomes scarce. Hence, the automotive industry has to adopt extensions like CAN FD or new communication protocols like Ethernet. In addition, the design and development in the automotive domain usually follows an iterative approach, where the existing configurations are usually inherited by later design series and only changed when necessary to save the cost for test and validation. Therefore, the resource-efficiency not only needs to be considered for the current design, but potentially also for future design [71]. A third reason is that there are usually some design conservativeness for safety purposes. Engineers for the function design and the system design need to take some assumption of the other side. In safety-critical applications, this assumption tends to be conservativeness so that the safety can be guaranteed [53, 73]. This design margin also consumes certain amount of embedded resources. Finally, as mentioned by some research works [54], sometimes some trade-off between different requirements can be made between the resource consumption and the functional performance. Therefore, when complying with the functional requirements, it would be possible to trade some functional performance with resource conservation.

The resource-efficient design also faces some challenges. The first challenge is due to the growing size and complexity of the system again, which makes the modeling and optimization more difficult. The other challenge is ensuring the safety of the system. When not adequate resources are provided because of resource conservation, the functional and non-functional correctness of the application can be jeopardized, thus leading to severe consequences for the safety-critical systems.

Flexible and Reconfigurable Design
Flexibility and adaptivity is another requirement that is becoming increasingly evident in the automotive domain. Traditionally, the configuration of the E/E system has been quite static. On the ECU side, the software components are compiled with the operating system into one single binary file and flashed onto the hardware and the software on most ECUs does not change during the entire life cycle of the vehicle. Similarly for the communication bus systems, the communication relationship between the ECUs and the mapping from signal-to-frame and frame-to-schedule- (or priority) are statically configured. The reason for this is that for safety-critical systems, the reduction of uncertainty contributes to the verifiability and determinism of the system [75].

However, this static paradigm might not be suitable for future requirements anymore. First of all, the rapid development in the ADAS and infotainment domain leads to increasingly more software functions developed in recent years. However, the design and development cycle of software is much shorter than the life cycle of a car. Therefore, the software functions of a car can easily become 'outdated'. It would be advantageous for the customer, if newly developed applications can be downloaded and installed on vehicles already sold. This ability is also known as PnP [4, 24, 76]. Furthermore, as the vehicle becomes increasingly software-intensive, the necessity of constant software updates has become obvious. These software updates can be used to fix flaws or add additional features. For example, Tesla has enabled the autonomous driving feature by remote software updates [44, 45]. To enable the software updates and PnP, the underlying system needs to possess certain level of flexibility and re-configurability, so that software can be updated, exchanged, installed and the necessary resources for them can be allocated and re-allocated. Secondly, there are increasingly more software applications that are multi-mode applications. Different operating modes might require different level of embedded resources. Towards resource-efficient design, it would be advantageous, if the embedded resources can be re-allocated. Both of the aforementioned aspects are reinforced by the increasing complexity and the ECU consolidation, where more applications are sharing the same resources.

There exist of course many challenges towards flexible and adaptive systems. The first one is the connectivity to the central servers, where the software can be downloaded. This challenges is currently being addressed by the development in the connected drive technologies, where increasingly more cars have access to this connection. The second challenge is the resource allocation problem. For PnP, software updates with changed resource requirements and multi-mode scenarios, the computation and communication resources need to be re-allocated online. One problem here is the online computation of the resource configurations. The other one is the ability of the system to re-allocate resources. Further challenges include the safety and security of the whole process. In terms of safety, the safe operation of the system during the re-configuration needs to be guaranteed. The security issue also plays an important role here since the connection to the outside opens up possible access of the attackers and during the reconfiguration process, the system could become vulnerable to security attackers.

Cross-Layer Design

Traditionally the design of automotive systems are relatively separate in different layers, the hardware, basis software, architecture and scheduling and application layer. With this paradigm the design of each layer has to make assumptions of the others. To guarantee the safety, such assumptions are usually quite conservative, therefore not leading to the most efficient utilization of resources. In recent years, there has been increasing effort in the cross-layer design of such systems. For example, hardware/software co-design has been one major direction [77], where the hardware and software system are co-designed according to different optimization objectives. Another emerging trend aligned with the CPS thinking is the control/platform co-design (sometimes also referred to control/architecture co-design) [52–56, 78, 79]. It has been identified that there is a strong interplay between the embedded systems and control design [53, 55, 56, 58]. This interplay can be exploited to reduce the design conservativeness while guaranteeing the correct functional behaviour by co-designing the control parameters and the embedded system parameters. It also offers design trade-offs between cross-layer objectives. One main challenge that needs to be addressed is the complexity problem. Many design methods like synthesis approaches are subject to the complexity problem, i.e., as the size of the system increases, the computational effort required grows drastically and beyond a certain point becomes not practical or event not tractable anymore. This problem is aggregated by the co-design methods in that they combine the dimensionality of two or more layers. A second challenge is the heterogeneity of the design and analysis methods across layers. For example, in the case of control/platform co-design, the set of design principles of the control engineering are quite different from those of embedded systems. Therefore, co-design methods have to deal with the inconsistency between these two sets of approaches, which adds to the complexity of the problem.

Further Requirements and Challenges

Besides the requirements mentioned above, there are certainly other requirements of importance. Since these requirements are out of the scope of this thesis, only a brief description is provided here. One example is the safety requirement. As increasingly more safety-critical functions are being deployed in modern vehicles, the correctness of these applications needs to be guaranteed, both the functional and non-functional correctness. Furthermore, fault-tolerance capability, especially fail-operational and fail-safe capability will be an essential topic for future autonomous driving vehicles [42, 80]. Another important requirement is security. As the modern vehicles are becoming increasingly softwarenized, the security problem has become a major topic [81]. Originally, the diagnosis port of the vehicle is considered to be the most vulnerable point. But as the vehicles are becoming increasingly networked, the security challenges have grown. On the other hand, the resources on the embedded systems are quite limited, therefore full-fledged security mechanism used in computer and network domain might not be applicable. Rather a light-weight security is desirable for automotive systems [82].

1.4 Scheduling and Schedule Synthesis

Scheduling is one important topic in real-time embedded systems and is the one that is most directly related to the real-time properties of the systems. The issue of scheduling arises from the problem of resource sharing and resource allocation. For example, on a processing unit, the computational resource can be translated to the computation time of software tasks and for a processing unit with a single core, only one task can access the computational resource at the same time. Similarly, when multiple messages are to be transmitted on a bus system and each message requires a certain transmission time, only one message can be sent on the bus simultaneous. To allocate the embedded resources like computation and communication, a certain scheduling scheme is necessary. The scheduling schemes on the processors and for the communication bus are quite diversified and depend on the corresponding OS or the bus protocols. However, in general, we could classify the scheduling schemes into two broad categories, namely the *time-triggered scheme* and the *event-triggered scheme.*

Time-Triggered and Event-Triggered Scheme
In the case of time-triggered scheme, the processes (e.g., software tasks and message transmission) are triggered by time-based schedules, which are usually implemented by periodical clock interrupts [83]. There are time-triggered schemes for both processor scheduling, e.g., OSEK-Time [38], and the bus scheduling, e.g., Time-Triggered Protocol (TTP) [84] and the FlexRay static segment [8, 9]. These schedules are usually computed offline and the operating systems or the bus protocols ensure that the schedules are kept at runtime. On the other hand, in the *event-triggered* scheme, the tasks and messages are triggered by events. Such events could be the production of a data, the finishing of a task or I/O interrupts. Examples of such scheduling schemes include the OSEK [38] on the processors and the CAN bus for communication.

Both scheduling schemes have their own advantages and disadvantages. The first advantage of time-triggered scheme is that the timing properties of the tasks and messages are more deterministic and thus providing the possibility of schedule optimization for different objectives (e.g., latency). Furthermore, the time-triggered scheme offers the easier composition of schedules. For example, if multiple sets of time-triggered schedules are to be integrated, the timing properties of each individual sets are not interfering with each other if there is no collision. In the case of collision, there exist approaches to integrate the schedules and resolve the collision [26, 85]. There are of course disadvantages of such scheduling schemes. The first one is the implementation overhead. For example, in a distributed time-triggered system, a synchronization protocol is required to synchronize the clocks of the network nodes and establish a concept of global time [83]. In general, this synchronization needs to be quite precise for the scheduling to be efficient. The synchronization requires also that the participating nodes are time-aware (e.g., FlexRay and TSN), which usually means more expensive hardware equipment. A second disadvantage of time-triggered scheduling is the lack of flexibility. Usually the schedules are pre-computed and static, and reconfiguration is not straightforward. In addition, the time-triggered scheduling often leads to less efficient utilization of resources, especially for messages of event-based nature. In comparison, the event-triggered scheduling scheme is more flexible, requires less overhead and makes more efficient utilization of the resources. In such a scheduling scheme, the task execution and message transmission is based on events and do not need to be confined to pre-defined time slots. When a task or message is added to the system or the resource requirement changes (e.g., execution time or transmission time), only analysis on timing and load needs to be performed to guarantee the schedulablity and the real-time properties, and if the requirements are meet, the configuration for the existing tasks and messages do not need to be re-computed. Secondly, event-triggered scheduling schemes requires less overhead. With few exceptions (e.g., FlexRay dynamic segment), the event-triggered systems do not need time synchronization since the global time is not needed on the system level. Finally, event-triggered schemes make more efficient utilization of the resources. For example, in the case of a task or message is not triggered, the corresponding computation time or bandwidth can be utilized by other tasks or messages, compared to the time-triggered scheme, where the specific time slots need to be reserved to certain tasks and messages even if they are not triggered. The first disadvantage of the event-triggered scheme is that it is not deterministic. In the best case, a worst-case timing guarantee can be provided by static analysis. However, this timing guarantee tends to be over-conservative and the scalability of the timing analysis techniques to larger systems is still a problem. The worst-case timing guarantee does not eliminate jitter, which is the timing difference between different instances of the task execution or message transmission. This jitter can have negative influence on the performance of the applications.

There exists, of course, scheduling schemes that contains a mixture of time-triggered and event-triggered schemes. Examples of this include the FlexRay protocol, commonly referred to as a hybrid protocol, and the TSN Ethernet protocol. The mixed scheduling scheme is often coupled with the mixed-criticality system, although the later term refers to the actual criticality of the applications, not on the scheduling scheme itself. For such systems using mixed scheduling schemes, one important problem is to reduce or eliminate the interference of the event-triggered part on the time-triggered part. This can usually achieved by time isolation. For example, the FlexRay protocol implements the two scheduling schemes on two different segments and TSN achieves this by implementing gate-open and gate-close events on the transmission control on the output ports of the switches [16]. In general, the applications tend to be of a mixed-criticality nature, where certain applications are more suitable using the time-triggered scheme and others using the event-triggered scheme. This mixed requirements can also happen within the same bus cluster. Therefore, hybrid protocols are gaining ground in the automotive domain.

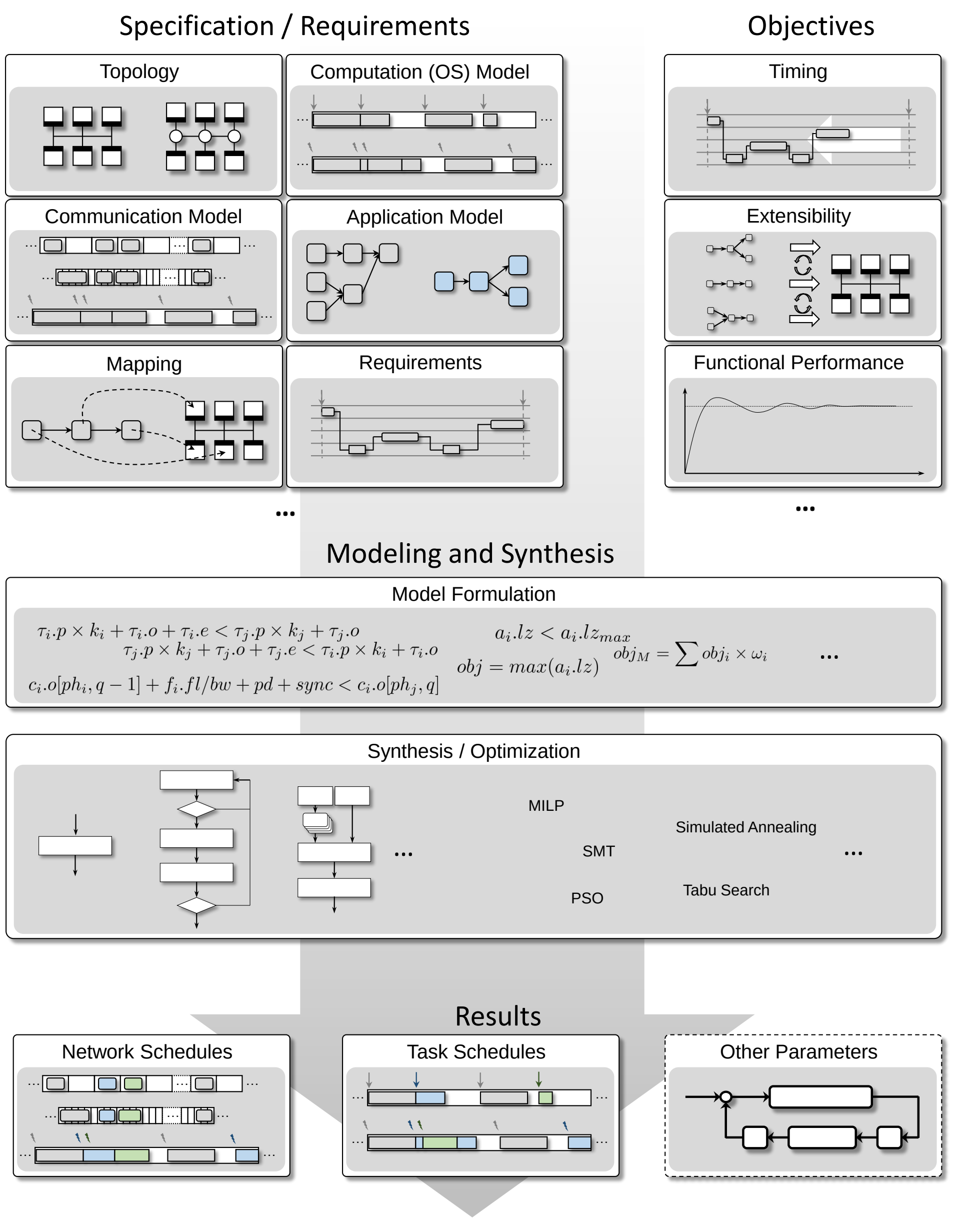

Figure 1.4: This figure shows the general flow of schedule synthesis approaches.

Schedule Synthesis

In the context of this thesis, applications with stringent real-time requirements like control applications are targeted. Therefore the primary concern of this thesis is the time-triggered scheduling scheme. For such a scheduling scheme, one essential issue that needs to be addressed is the computation of the time-triggered schedules. Such schedules can be computed manually or in an ad hoc fashion. However, the problem is that this process is tedious, error-prone and easily becomes intractable when the system gets large. Therefore, the more efficient way is to automatically *synthesize* the schedules.

Figure 1.4 shows a schematic for the schedule synthesis methods. The prerequisite for the schedule synthesis is efficient and precise modeling of the system. On one hand, the design specification, e.g., the hardware topology, scheduling of the OS and communication protocols, the applications, mapping, and requirements like timing requirements are modeled. On the other hand, optional optimization objectives can be modeled. Based on the models, the constraints of the problem can be formulated and the schedules are the design parameters to be determined, where the design space is specified by the constraints. The whole problem can be formulated into a mathematical programming (e.g., Mixed Integer Linear Program (MILP)) or an Satisfiability Modulo Theories (SMT) problem and be solved with specific solvers. Often the constraints are difficult for a closed-form mathematical formulation or the formulated problem is too complex to solve. In this case, meta-heuristic or heuristic methods can be employed. The synthesis problem can also be divided into different iterations or some pre-design can be done to prune the design space. Furthermore, for optimization problems, which is more computational expensive than a synthesis problem without optimization, trade-offs between the optimality and computational effort can be made. If the models are accurate and the constraint formulations are correct, the resulted schedules are guaranteed to be safe, i.e., meeting the system constraints and design requirements.

The main advantage of the synthesis approaches is the automated design and the possibility for systematic optimization. Modern vehicles contain increasingly more software components and thousands of signals (see Figure 1.3). Although the design can usually be broken done into functional domains or subsystems, the problem is still too complex for manual design. The schedule synthesis approaches relieve the engineers from tedious and error-prone manual design and the results are also guaranteed to be safe, provided the modeling is correct. Design optimization is also an important issue in current and future systems with diverse requirements, e.g., on the real-time capability, reliability, safety, etc. Schedule synthesis methods offer the possibility for systematic optimization and even trade-offs between different optimization objectives [54].

There have also been some recent development on this topic. One trend is the co-synthesis of schedules and parameters on other layers, e.g., on the functional layer. Towards this, there have been considerable effort in control/platform co-design, which synthesizes jointly the schedules and the control parameters. There exist also other methods combining the schedule synthesis with other layers [77]. Another emerging trend is the online schedule synthesis. Schedule synthesis is usually done offline during design and development phase.

However, as already mentioned in the current and future trends, the requirement on the flexibility of the system increases. Towards this requirement, there has been increasing effort to address the online schedule synthesis problem. Such effort targets at the ability of automotive systems to dynamically adapt the schedules and resource allocation during run-time to accommodate newly installed applications, software updates or multi-mode applications. However, for both of these directions, there are still considerable hurdles to be overcome.

Real-Time Systems

A real-time system differs from the non-real-time system by the importance of time. As defined in [83], no only the correctness of the logical computations, but also the time when the results are produced matter for the correctness of the system behavior in a real-time system. In the embedded systems community, the real-time requirement is usually defined by the *deadline*, a time where a specific process needs to be completed or a result needs to be produced. The deadline can also be classified into *soft*, *firm* and *hard* deadline [83]. In the case of soft deadline, the result will still be useful if the deadline is missed. Otherwise the deadline is firm. If severe consequences can happen by missing a firm deadline, it is referred to as a hard deadline. If a system has at lease one hard deadline, then it is a *hard real-time system*, otherwise it is a *soft real-time system* [83]. In general, a deadline miss in a soft real-time system could lead to deterioration of the desired performance and a deadline miss in a hard real-time system could lead to catastrophic results.

There are different timing properties for real-time system: (i) *Task response time:* This refers to the time between the release of a task, i.e., the task is ready to run, and the completion of the execution of the task. (ii) *Message response time:* Similar to the task response time, the message response time denotes the time between the time when the message is ready to be transmitted and the time when the transmission finishes. (iii) *End-to-end timing:* A distributed application might consists of several tasks with data dependency and these data are sent on the network as messages. The end-to-end timing is determined by the interplay between the tasks and messages. For example, the end-to-end latency (or end-to-end delay) refers to the time between the start of the execution of the first task in a chain and the completion of the last task in the chain. In the case of an application consisting of tasks of an acyclic graph, the end-to-end latency has to be defined according to specific requirements. The end-to-end latency is particularly important for certain application like feedback control loops. To achieve a low end-to-end latency, low response time for the tasks and messages as well as the synchronicity between them are important. Sometimes the end-to-end response time is also important to applications that need to be finished as soon as possible. This refers to the time between the release time of the first task and the completion of the last task. (iv) *Jitter:* Jitter is also an important timing property and generally refers to the difference between the timing between two different instances of a task or message. Besides the latency and response time, jitter sometimes also plays an important role in the timing requirements of applications.

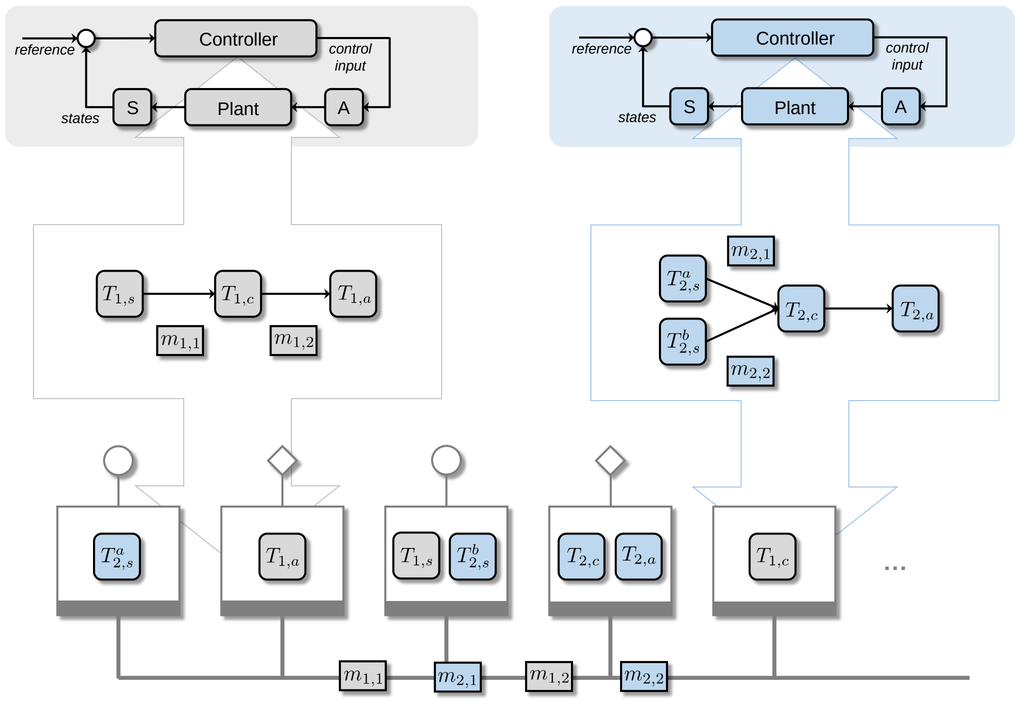

Figure 1.5: This figure shows a schematic example of the automotive CPS. The software implementation of the control system is partitioned and mapped on the distributed architecture.

1.5 Design of CPS

CPS is a concept that has emerged in the past decade [48–50], as a result of increasingly more software implementation of embedded control systems in industrial applications, the increasing size of such systems and the tighter interaction between the computational units and the physical processes to be controlled. The design of CPS involves mainly two communities, namely the control and the embedded systems community. The control engineers design the controller on the application level, while the embedded systems engineers design the system level parameters like the software partition, mapping and scheduling. As the size of the systems increases, both communities have become aware of the importance of interaction and cooperation.

Embedded Control Systems

Embedded control systems are usually implemented as software running on processors. The feedback control system is a common form of control systems. Such a system consists in general of three logical components, namely the *sensor*, the *controller* and the *actuator*. The sensor measures the measurable states of the plant, i.e., the physical process to be controlled. The controller computes the controller input and the actuator exerts this on the physical system. In an embedded control system, the sensor, controller and actuator functions are implemented as software tasks. In a distributed case, the software tasks could be mapped on different ECUs and the data between them, e.g., sensor data or control input, are sent on the communication buses. Figure 1.5 shows a schematic representation of a CPS. This implementation has several constraints on the control system. Firstly, software tasks take some time to execute, and for sophisticated controllers, this time is often not negligible. In addition, the communication and the interplay between the tasks and messages introduce extra time. This means that there is often a non-negligible delay between the sensing process and the actuation process and this delay could be subject to jitter. Secondly, there is usually limited resources on embedded systems, not enough to run the tasks as frequent as possible. This means that the controller action can only be triggered at certain period, referred to as *sampling period.* Common control applications are triggered periodically, but there are also controllers that are not triggered periodically, e.g., in event-triggered control [86]. It has been identified, that the sampling period and the delay have considerable influence on the control performance of the system. Conventionally, the controller and the embedded system are designed separately by engineers of different expertise and background. In this case, both sides have to make some assumptions on the other side. For control applications, which are usually safety-critical, this assumption tends to be conservative [53, 73]. As a result, the system design might not be optimized in terms of control performance or resource efficiency.

Control/Platform Co-Design

Over the years, both the control and embedded systems communities have made efforts on exploiting the interaction between the controller design and platform design to enhance the efficiency of the designed systems. A group of works that have been drawing increasing attention recently is the control/platform co-design [54, 55, 78, 87]. These works consider both the controller design and the platform design as a holistic problem and simultaneous synthesize the control and platform parameters. The design process is similar to that of schedule synthesis shown in Figure 1.4. The difference is that besides the specification and objectives for the platform design, those for the control design is also considered. The specification of control design includes, e.g., the model of system dynamics, controller type, control stability and requirements for the control performance. The objectives for the controller design can be the control performance of one or multiple control metrics. Once these models are obtained, they are formulated into a co-synthesis or co-optimization problem and solved. Due to the different nature of modeling in control and platform, it is often difficult to find a closed-form formulation for the combined problem. Therefore, control/platform co-design methods usually employ some heuristic methods and (or) staged optimization methods, e.g., an iterative approach [78, 79] or some pre-design steps [54]. The result of the co-design are the platform and control parameters, possibly jointly optimized towards certain objectives. The advantages of such co-design approaches are automated design and optimized results. The automated design of both the control and platform parameters can relieve engineers from tedious and costly process of testing and integration of both controllers and their implementation. Secondly, the control/platform co-design can reduce the conservativeness in the separate design process and therefore achieving more efficient design. It also allows the optimization of the parameters towards certain objectives and even offer engineers the design trade-off between different objectives. The challenges for the control/platform co-design, as already mentioned in the explanation on cross-layer design, are the complexity and heterogeneity problem. The combination of the design space of both the control and platform design considerably increases the complexity of the design/synthesis problem, thus limiting the scalability of such approaches. In addition, the co-design methods have to deal with a heterogeneous design space, i.e, the platform and control design employ completely different methods and paradigms. Therefore, inconsistency in modeling and design methods introduces some challenge in bringing both sides under one single problem. Besides these two, an extra challenge is to deal with dynamic behaviour of such systems. Existing co-design methods address mainly the static design of CPS. The co-design enabling dynamic behaviour, e.g., switched control systems and adaptive platform, has not yet been sufficiently addressed, although there have been some initial efforts on this [56].

1.6 Thesis Contributions

As discussed previously in this chapter, the increasing number of software-based applications mapping on the E/E architecture has resulted in the increasing size and complexity. This has introduced challenges for meeting traditional requirements on real-time capability, resource-efficiency and emerging requirements on flexibility, adaptivity, safety and security. Towards this, this thesis proposes three new approaches to address respectively the requirements of real-time capability, flexibility/reconfigurability and resource-efficiency and cover emerging topics of automotive Ethernet and CPS. The main contributions of this thesis are listed below.

- The first contribution is an approach to co-synthesize task and communication schedules for an Ethernet-based time-triggered system.

  Many of the applications in the automotive domain are safety-critical and time-critical applications. These applications have stringent requirements on the timing properties like latency and jitter. Moreover, for most distributed applications, it is the application-level timing (or end-to-end timing) that really matters for the safety and performance of the application. Ethernet has recently emerged as a promising candidate for the next generation in-vehicle communication protocol. To fulfill the real-time requirements, some Ethernet protocols like TSN and AS6802 implement time-triggered traffic to offer deterministic communication.

  The proposed approach formulates the schedule synthesis and optimization problem into a Mixed Integer Program (MIP) problem. It specifies the constraints for network communication, task scheduling as well as the interaction between the tasks and communication. It further considers the switched Ethernet topology and specific timing properties like the synchronization precision. In addition, application-level timing objectives are also formulated and a multi-objective optimization method is proposed that can handle different timing objectives and their combinations. The proposed method is independent of task and communication configurations as well as network topologies and device performance parameters. The experimental results show the applicability of the method towards synthesizing combined task and network schedule according to timing objectives and that it can be scaled to system of reasonably large size.

- The second contribution is a schedule management framework for the case of PnP and software updates in cloud-based future automotive software systems.

  The development in the ADAS and infotainment domain has led to the development of increasingly more new software applications. When the car is moving towards autonomous driving, this trend is likely to accelerate. However, the design and development cycle of software and electronics is much shorter than the life cycle of a car. Therefore, there have been emerging requirements on the update of software and the PnP of new applications. To allow this, the platform needs to possess a certain level of flexibility. One important issue here is the allocation of communication and computation resources. In the case of time-triggered systems, the schedules need to be obtained online.

  Towards this, this thesis proposes a schedule management framework to generate and manage time-triggered schedules for the PnP and software update scenario. This software framework is based on a client-server architecture and utilizes both the computation and storage resource onboard the vehicle and in backend servers using the connectivity of the vehicle to the cloud. Each side contains a management module, a synthesis module, a web module and a configuration pool. The synthesis module is used to synthesize schedules online. The web module communicates the client and the server and the management module oversees the whole process. The configuration pool is used to store schedule configurations to facilitate their reuse and reduce synthesis efforts. In the web module, websocket secure is used to allow bilateral communication and offer secure communication. When a reconfiguration request comes, the client starts its own process while sending the request to the server. Both sides search for valid configuration in the configuration pool and if such a configuration can not be found, a synthesis process is started to compute schedules. The fastest result, either from the client or the server, will be taken. In the synthesis process, a four-staged strategy is proposed to make a trade-off between the synthesis time and disturbance to existing applications on one side and the chance of accommodating new applications on the other. The four stages range from incremental design to complete re-synthesis and gradually increase the number of applications to be re-synthesized. The proposed framework is prototypically implemented on a Raspberry

Pi as the client and a notebook computer as the server. The experiment shows that the framework can efficiently generate and manage schedules online and the combined usage of both onboard and cloud-computing efficiently reduces the schedule synthesis time. Furthermore, the four-staged strategy offers a transition from low synthesis time and disturbance to existing applications to higher chances of accommodating new applications.

- A further contribution is a scheme for resource-efficient implementation of CPS based on hybrid communication protocols.

  Efficient utilization of resource is an important design requirement. Hybrid protocols like FlexRay offer both Time-Triggered (TT) and Event-Triggered (ET) communication. The TT communication offers higher timing predictability, but the amount of such resources are often quite limited and expensive due to various reasons. By allowing a control application to switch between TT and ET communication, valuable TT resource can be shared between different control applications thus conserving TT resource while achieving better control performance than pure ET based implementation.

  The scheme proposed in this thesis follows this switching paradigm. However, the proposed scheme allows a control application only to use as much TT resource as meaningful and switch it back to ET resource as soon as possible, possibly before it is settled. This allows the released TT resource to be utilized by other control applications as soon as possible so that the overall control performance of the system is enhanced. Firstly, an automated control design method is proposed to synthesize controllers for both the TT and ET case jointly so that the combined performance is optimized and the switching stability between the two cases is guaranteed. The whole process employs a Particle Swarm Optimization (PSO) based method for performance optimization and uses a Linear Matrix Inequity (LMI) based feasibility check. When the controllers for both cases are designed, a table containing meaningful switching sequences is generated for each control application. The table contains the minimal number of switching sequences that can be used to predict the control performance given a particular situation. At run-time, when a disturbance arrives, an online scheduling algorithm is executed to compute the allocation of the TT resource and the switching sequences of the control applications. This online algorithm is based on the control performance prediction using the table of switching sequences and computes the current best solution according to the overall control performance. Furthermore, the challenge of implementing the switching between TT and ET communication compliant to the FlexRay protocol and Commercial-Off-The-Shelf (COTS) tools is addressed by a middleware based solution. The results show that the proposed scheme can offer a trade-off between the utilization of TT communication resource and the control performance. Compared to existing switching schemes, the proposed scheme can achieve better control performance with the same resource utilization.

1.7 Organization and Publications

1.7.1 Organization

This thesis is organized in the following way. This chapter provides the basic introduction to the context of this thesis, where the motivation, background, current and future trends as well as the requirements and challenges are explained. From Chapter 2 to Chapter 4, three new approaches addressing the challenges mentioned are presented. In Chapter 2, a method on the co-synthesis of task- and network schedules for the Ethernet-based time-triggered system is presented. This is followed by the description of a proposed framework for the schedule management framework for the cloud-based future automotive software systems in Chapter 3. Chapter 4 then presents a scheme for the resource-efficient implementation of a resource-efficient CPS based on the hybrid communication protocols. Finally in Chapter 5, the concluding remarks as well as the future outlook are provided.

1.7.2 Publications

The contributions presented in Chapter 2 are closely related to the following publications and mainly appeared in (1):

(1) Licong Zhang, Dip Goswami, Reinhard Schneider, Samarjit Chakraborty, *Task- and Network-Level Schedule Co-Synthesis of Ethernet-Based Time-Triggered Systems*, in Proc. of Asia and South Pacific Design Automation Conference (ASP-DAC), 2014.

(2) Reinhard Scheider, Licong Zhang, Dip Goswami, Alejandro Masrur, Samarjit Chakraborty, *Compositional Analysis of Switched Ethernet Topologies*, in Proc. of Design, Automation and Test in Europe (DATE), 2013.

(3) Licong Zhang, Reinhard Schneider, Alejandro Masrur, Martin Becker, Martin Geier, Samarjit Chakraborty, *Timing Challenges in Automotive Software Architectures*, in International Conference on Software Engineering (ICSE) Companion, 2013.

The contributions presented in Chapter 3 are closely related to the following publications and mainly appeared in (4):

(4) Licong Zhang, Debayan Roy, Philipp Mundhenk, Samarjit Chakraborty, *Schedule Management Framework for Cloud-Based Future Automotive Software Systems*, in Proc. of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2016.

(5) Philipp Mundhenk, Ghizlane Tibba, Licong Zhang, Felix Reimann, Debayan Roy, Samarjit Chakraborty, *Dynamic Platforms for Uncertainty Management in Future Automotive E/E Architectures*, in Proc. of Design Automation Conference (DAC), 2017.

The approaches presented in Chapter 4 are closely related to the following publications:

(6) Diptesh Majumdar, Licong Zhang, Purandar Bhaduri, Samarjit Chakraborty, *Reconfigurable Communication Middleware for FlexRay-Based Distributed Embedded Systems*, in Proc. of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2015.

(7) Debayan Roy, Licong Zhang, Wanli Chang, Dip Goswami, Samarjit Chakraborty, *Multi-Objective Co-Optimization of FlexRay-Based Distributed Control Systems*, in Proc. of IEEE Real-Time and Embedded Technology and Applications Synposium (RTAS), 2016.

The following publications are not directly part of this thesis, but are generally on the topic of automotive CPS and thus related to this thesis:

(8) Debayan Roy, Licong Zhang, Samarjit Chakraborty, *Automated Synthesis of Cyber-Physical Systems from Joint Controller/Architecture Specifications*, in Proc. of IEEE International Forum on Specification and Design Languages (FDL), 2016.

(9) Wanli Chang, Debayan Roy, Licong Zhang, Samarjit Chakraborty, *Model-based Design of Resource-Efficient Automotive Control Software*, in Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2016.

(10) Ghizlane Tibba, Christoph Malz, Christoph Stoermer, Natarajan Nagarajan, Licong Zhang, Samarjit Chakraborty, *Testing Automotive Embedded Systems under X-in-the-Loop Setups*, in Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2016.

(11) Qi Zhu, Hengyi Liang, Licong Zhang, Debayan Roy, Wenchao Li, Samarjit Chakraborty, *Extensibility-Driven Automotive In-Vehicle Architecture Design*, in Proc. of Design Automation Conference (DAC), 2017.

(12) Ramesh S, Birgit Vogel-Heuser, Wanli Chang, Debayan Roy, Licong Zhang, Samarjit Chakraborty, *Specification, Verification and Design of Evolving Automotive Software*, in Proc. of Design Automation Conference (DAC), 2017.

The following publications are not directly related to the topic of this thesis, but the works fall in the broad domain of CPS:

(13) Sangyoung Park, Licong Zhang, Samarjit Chakraborty, *Design Space Exploration of Drone Infrastructure for Large-Scale Delivery Services*, in Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2016.

(14) Sangyoung Park, Licong Zhang, Samarjit Chakraborty, *Battery Assignment and Scheduling for Drone Delivery Bussinesses*, in Proc. of IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), 2016.

Chapter 2

Schedule Co-Synthesis for Ethernet-based Time-Triggered Systems

This chapter introduces an approach for the schedule synthesis for the Ethernet-based time-triggered systems. In particular, it addresses the problem of co-synthesizing task and communication schedules according to application-level timing objectives. As mentioned in Chapter 1, some of the applications in the automotive domain are safety-critical and time-critical functions, which have very stringent timing requirements on the properties like latency, jitter. Many of such applications are distributed, where the dependent software tasks belonging to the same application are mapped on different nodes on the network and the data between them are transmitted over the communication network. For such applications, the typical scheduling scheme used is the time-triggered scheme. Recently, Ethernet-based protocols are gaining ground in industrial domains, e.g., AS6802 standard or the TSN standards. Such protocols offer support for time-triggered traffic in the Ethernet network in a mixed-criticality fashion. Considering the fact that for many distributed applications like the feedback control loop, it is the application-level timing that really matters, the approach presented in this chapter combines the task scheduling and communication scheduling in a synchronized manner and optimizes schedules according to application-level timing objectives. The approach further takes into account a number of Ethernet-specific timing parameters such as interframe gap and synchronization precision. The scheduling synthesis problem is formulated as a MIP problem. The approach is able to handle one or multiple timing objectives such as application response time, end-to-end delay and their combinations. The experimental results show the applicability of the proposed approach using an industrial size case study and further show that the approach scales to systems with reasonably large size.

Chapter outline: This chapter is divided into six sections. A general introduction into the context is provided in Section 2.1, where the motivation, the overview and the contributions of this work are explained. In Section 2.2, the related work is reviewed. In Section 2.3, the setup considered in this chapter, an Ethernet-based time-triggered system, is presented. This is followed by the proposed approach in Section 2.4, the major part of which is the formulation of the constraints and the objectives in the schedule synthesis problem of such a system. The experimental results and evaluations are presented in Section 2.5 before the concluding remarks in Section 2.6.

2.1 Introduction

Recently, several Ethernet based protocols are developed and adopted in real-time safety-critical domains. The examples are EtherCAT and PROFINET [18] in industrial automation, AFDX [20] and AS6802 [21] in avionics and the AVB networks [14]. The majority of these protocols are based on the original Ethernet standard IEEE802.3 and employ a *switched* network paradigm. That is, switches are used to connect the *end stations* via *full-duplex* links. A full-duplex link consists of two independent directed links and allows simultaneous transmission in both directions. The upper part of Figure 2.1 shows an example of such a network with four end stations and two switches.

In an Ethernet-based network, the end stations communicate by exchanging messages as Ethernet *frames*. The frames are forwarded link by link via the switches from the sender to receiver end station. In general, when the frames are forwarded to the same link simultaneously, they are stored in a queue and sent according to an output scheduler (e.g., strict priority), as shown in Figure 2.2. Thus, the queueing in switches results in a queueing delay which further depends on many factors such as the topology, traffic load and often introduces a considerably large delay. Although there exist several approaches that provide analytical bounds for the worst-case latency [62–64], this sort of network still suffers from non-deterministic temporal behaviour and it is difficult to achieve very low latencies. This makes the traditional Ethernet-based networks unsuitable for safety-critical applications with stringent timing requirements. To address this problem, most of such protocols offer mixed-criticality services by dividing the traffic into different classes, applied to communication with different timing requirements. For example, PROFINET has Isochronous Real-Time (IRT), Real-Time (RT) and Non-Real-Time (NRT) services in decreasing timing guarantee. Similarly, the traffic in AS6802 is divided into TT, Rate-Constrained (RC) and Best-Effort (BE) traffic and in AVB networks Class A/Class B are differentiated with normal prioritized frames.

In the above context, the traffic class designed to provide most strict timing guarantee

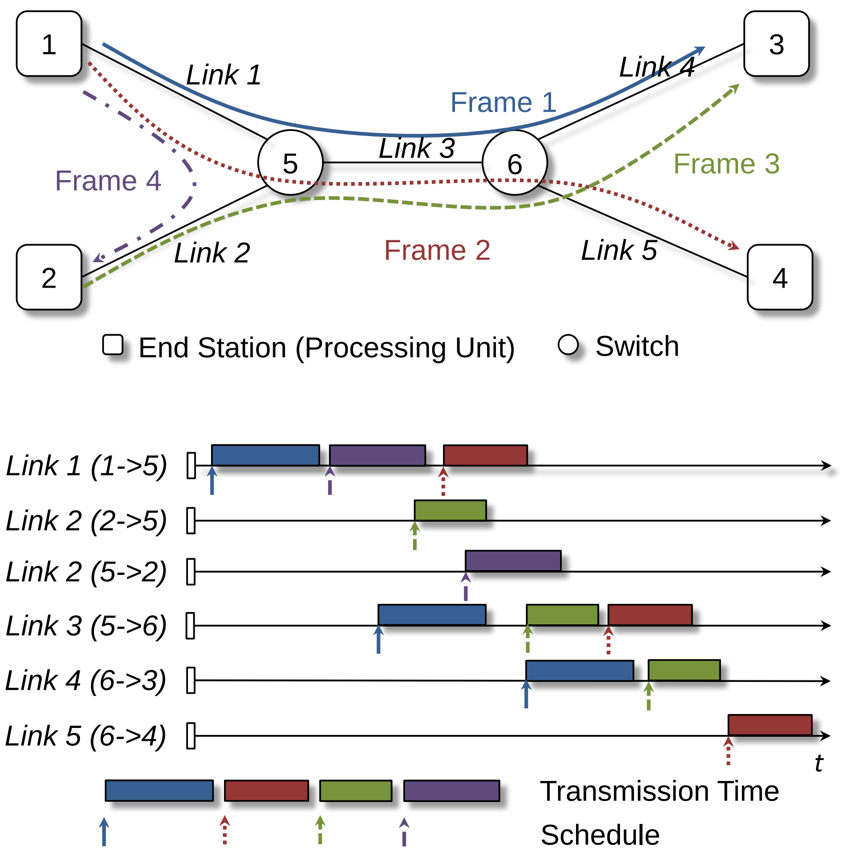

Figure 2.1: An example of time-triggered traffic in switched Ethernet network.

is the time-triggered traffic. Examples of this traffic class include the IRT in PROFINET, the TT traffic in AS6802 and the time-triggered traffic in the IEEE802.1Qbv standard [16]. The actual implementation of the time-triggered traffic class may vary from one protocol to another, but the basic idea is to schedule the frames so that the delays introduced by queueing in switches can be minimized or eliminated, thus achieving low network latency and jitter. Certain mechanisms are used to remove the influence from other traffic classes so that the frame transmission follows the pre-defined schedule. For example, PROFINET uses a dedicated time period in a cycle to transmit IRT frames. All the frames in this traffic class are sent and forwarded in the network according to a pre-configured schedule. A schedule defines the exact time point when the end stations and switches send the frames on the Ethernet links. As shown in Figure 2.1, the schedules should be chosen such that the frame transmission times do not collide with each other and thus ensuring deterministic temporal behavior with low latency and jitter. In this work, we consider the time-triggered traffic. Communication of other traffic classes can be added to the system without influencing the timing properties of time-triggered communication. We present our method considering the time-triggered traffic in a general switched Ethernet network. Our method

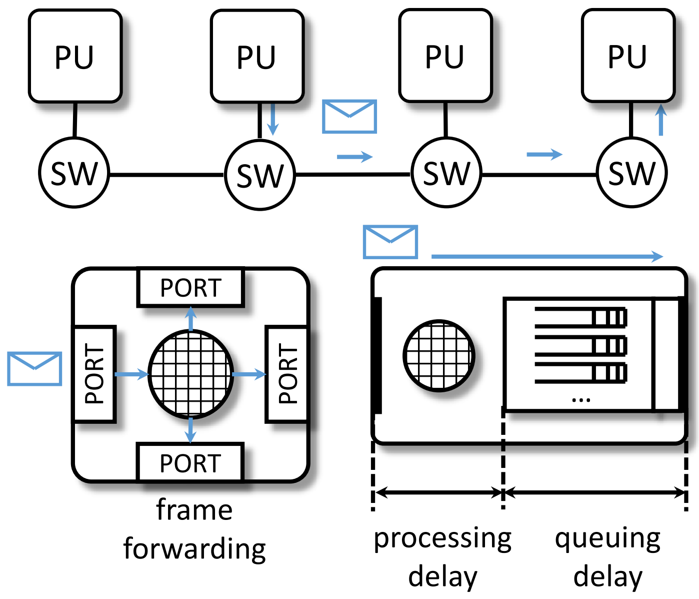

Figure 2.2: This figure shows a simplified schematic for the frame forwarding and the delay for a priority-based switched Ethernet network. PU and SW refer respectively to processing unit and switch.

can be tailored to fit into specific protocols such as PROFINET, AS6802 or TSN.

In a distributed system, a number of functions (or applications) are performed by a combination of *tasks* running on the end stations and data exchange by the communication over the network. For example, a distributed control system might have its *sensing*, *computing* and *actuating* tasks mapped onto different end stations and sensing/control data is sent via the network. In such cases, the performance of an application depends on schedules of both tasks and communication. Since optimal communication schedules may not necessarily result in optimal schedules at the application-level, the above problem can not be simplified to that of the communication schedules alone. Figure 2.3 shows an example of the comparison between synchronized and unsynchronized task and network schedules and the resulting end-to-end latency. Moreover, the representation of design objectives is often itself a problem. For example, the design objective can be to minimize the overall latency of all applications while placing emphasis on certain ones. Such design requirements need multi-objective formulation. In this work, we aim to optimize the schedules for the tasks running on the end stations and for communication over the network considering one or multiple optimization objectives.

Contributions: The method proposed here is for the application-level schedule synthesis of distributed systems using time-triggered traffic in Ethernet communication. In this approach, the scheduling problem is formulated as a Mixed Integer Programming (MIP) model. Furthermore, different optimization objectives are considered and a multi-objective

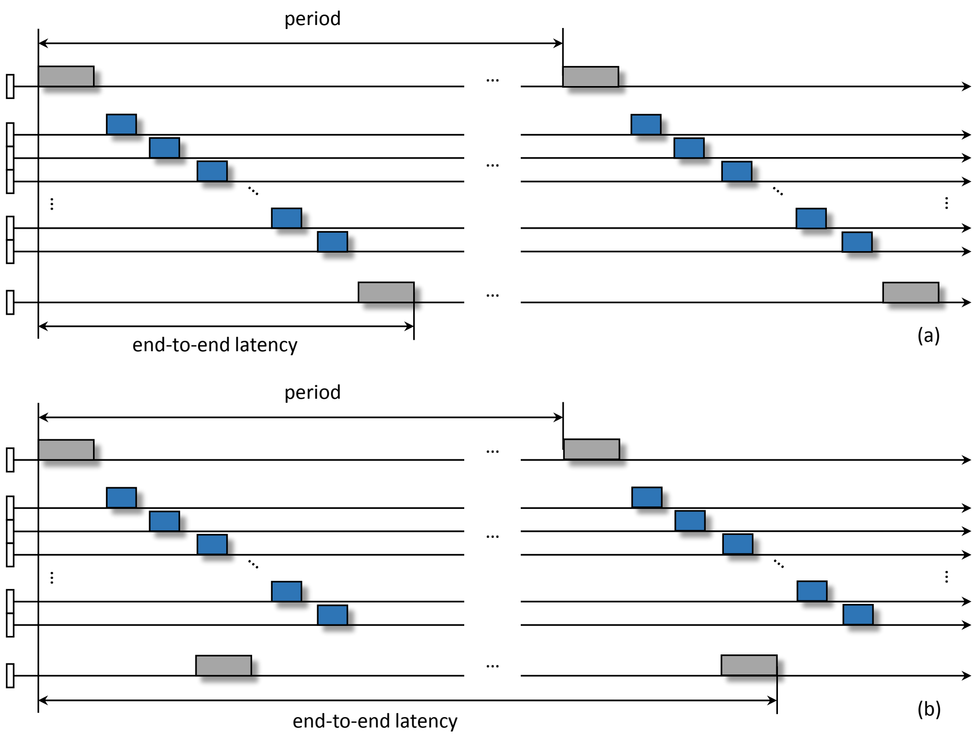

Figure 2.3: Difference between (a) a synchronous task and network schedule and (b) an asynchronous task and network schedule.

optimization problem formulated. We use Gurobi [88] to obtain a solution for the MIP formulation. Although there are a few recent works [59, 89] towards optimizing communication schedules in this context, our work focuses on application-level optimization. Using an industrial size case study, we illustrate various design aspects of such Ethernet-based time-triggered systems by considering different optimization objectives. In addition, we have also introduced generalized constraint and objective formulations for the end-to-end timing. To the best of our knowledge, this is the first work to consider the interplay between the task and communication schedules of a time-triggered Ethernet-based distributed system and optimize the schedules according to the (multi-)objectives at the application-level.

2.2 Related Work

The research works that are related to this context can be divided into three groups.

(i) Firstly, there have been some works on the general time-triggered architecture [83, 90, 91]. These works introduced the time-triggered architecture and presented general techniques to deal with with time-triggered systems without addressing the actual protocols.

(ii) There have also been many research works on the schedule synthesis for FlexRay-based systems [55, 68, 69, 92]. These works deal with both network [68] and application level schedules [55, 69, 92]. However, FlexRay as a communication protocol differs considerably from a switched Ethernet network and works in [55, 68, 69] can not be applied in a straightforward manner.

(iii) The third group of related works address the synthesis and optimization problem in time-triggered Ethernet networks [59, 89, 93, 94]. Towards this, [93] proposed a model formulation for the schedule synthesis for the PROFINET IRT which is the time-triggered traffic pattern in PROFINET. Similarly, [59] formulated the complete model of the time-triggered traffic in TTEthernet (AS6802) in SMT, which is later complemented by [89] for the bandwidth reservation of RC traffic. Further, [94] proposed an alternative schedule synthesis method of TTEthernet based on Tabu search, such that the deadlines of TT and RC messages are satisfied and end-to-end delays of RC messages are minimized. These approaches, however, focus mainly on the schedule synthesis for the network. As already explained in Section 2.1, for the distributed applications, it is usually the application-level timing that is important. Compared to these related works, this approach at hand considers both the task and network scheduling and complex application-level timing objectives.

2.3 Problem Formulation

We consider a distributed system whose physical topology consists of a number of *end stations* connected by a switched Ethernet network. We denote the system as a graph $G(\mathcal{V}, \mathcal{E})$, where vertices $\mathcal{V}$ denote the set of network *nodes* (end stations and switches) and edges $\mathcal{E}$ denote the *full-duplex* Ethernet links. To differentiate between different types of nodes, we denote an end station as $v_i^e \in \mathcal{V}$ and a switch as $v_i^s \in \mathcal{V}$. A full-duplex link connecting two nodes v_m and v_n is denoted by $l_{m,n} \in \mathcal{E}$ and $l_{n,m} \in \mathcal{E}$, each representing a *directed* link from v_m to v_n and from v_n to v_m. In the distributed setup under consideration, we consider the following components which are described using an example shown in Figure 2.4.

Application tasks: We define a task running on an end station as an application task. We consider periodic application tasks τ_i which are characterized by a tuple $\tau_i = \{\tau_i.p,\ \tau_i.o,\ \tau_i.e\}$ consisting of the period, offset and the Worst-Case Execution Time (WCET) respectively. In the example shown in Figure 2.4, there are five applications tasks τ_1 to τ_5, which are running in five different end stations.

Communication tasks: $c_i = \{f_i,\ c_i.tr,\ c_i.o,\ c_i.p\}$ can be used to define a communication task c_i. Each communication task is associated with an unique Ethernet frame with frame length $f_i.fl$. We define a *path* as a route from the sender to one receiver. For example, in Figure 2.5, $c_1.ph_1$ is one path from v_1^e to v_3^e. Further, $c_i.tr$ denotes the *path tree* which consists of all the *paths* from a particular sender station to the receiver stations. A path tree may contain one path $c_i.tr = c_i.ph_1$ or multiple paths $c_i.tr = \{c_i.ph_1,\ c_i.ph_2,\ \dots,\ c_i.ph_{\alpha_i}\}$, depending on whether the uni-cast or multi-cast operation is applied. For example, in

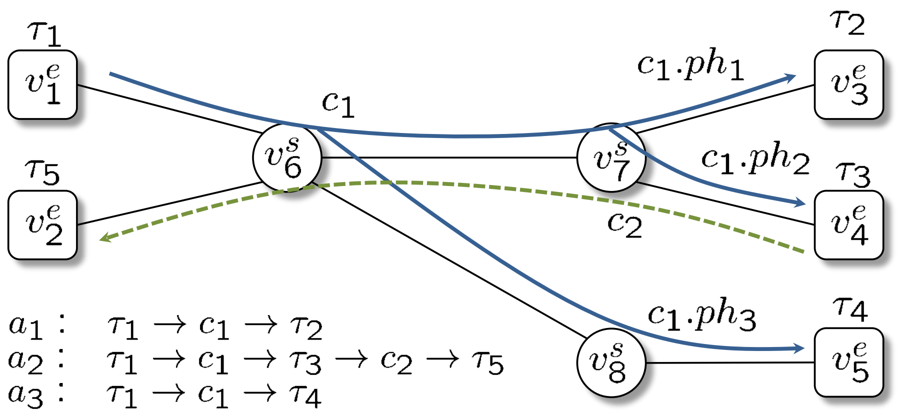
τ_1
v_1^e
τ_5
v_2^e
c_1
v_6^s
v_7^s
v_8^s
$c_1.ph_1$
$c_1.ph_2$
$c_1.ph_3$
c_2
τ_2
v_3^e
τ_3
v_4^e
τ_4
v_5^e
$a_1: \tau_1 \rightarrow c_1 \rightarrow \tau_2$
$a_2: \tau_1 \rightarrow c_1 \rightarrow \tau_3 \rightarrow c_2 \rightarrow \tau_5$
$a_3: \tau_1 \rightarrow c_1 \rightarrow \tau_4$

Figure 2.4: An example of a distributed system of applications a_1 to a_3, application tasks τ_1 to τ_5 and communication tasks c_1 and c_2. This figure shows the task mapping and task chains of applications.

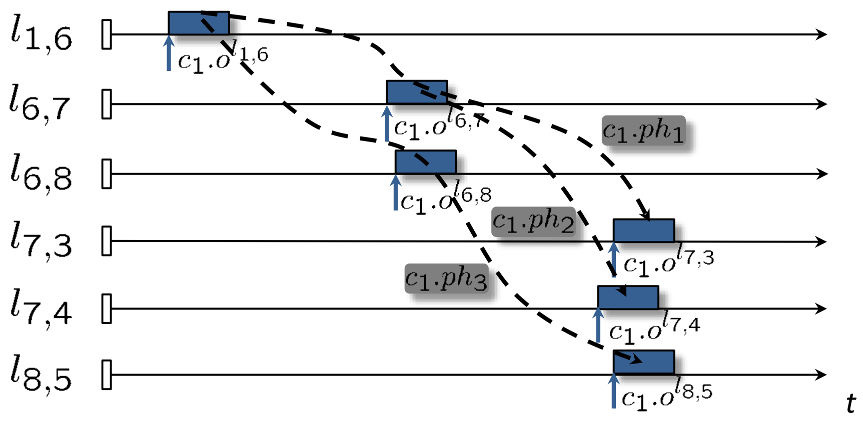

Figure 2.5: This figure shows the path tree and schedules of the communication task c_1 of the example in Figure 2.4.

Figure 2.4 and Figure 2.5, path tree of c_1 contains three paths $c_1.ph_1$, $c_1.ph_2$ and $c_1.ph_3$. Furthermore, $c_i.o$ denotes the set of sending schedules of frame f_i on all the Ethernet links in the path tree. A single schedule on link $l_{m,n}$ is represented as $c_i.o^{l_{m,n}}$. In Figure 2.5, the communication task c_1 involves six Ethernet links and there is one schedule for each link as indicated in Figure 2.5. Finally, $c_i.p$ denotes the period of the communication task. Note that as oppose to the traditional task, the communication task here represents the whole process of sending, forwarding and receiving a frame over the network.

Applications: An *application* a_i is a collection of certain application tasks and communication tasks that perform an independent function. An application is characterized by the tuple $a_i = \{a_i.tc,\ a_i.p,\ a_i.rt,\ a_i.lz\}$. Here, $a_i.tc$ is the *task chain*, containing the application and communication tasks that constitute the application in the temporal order.

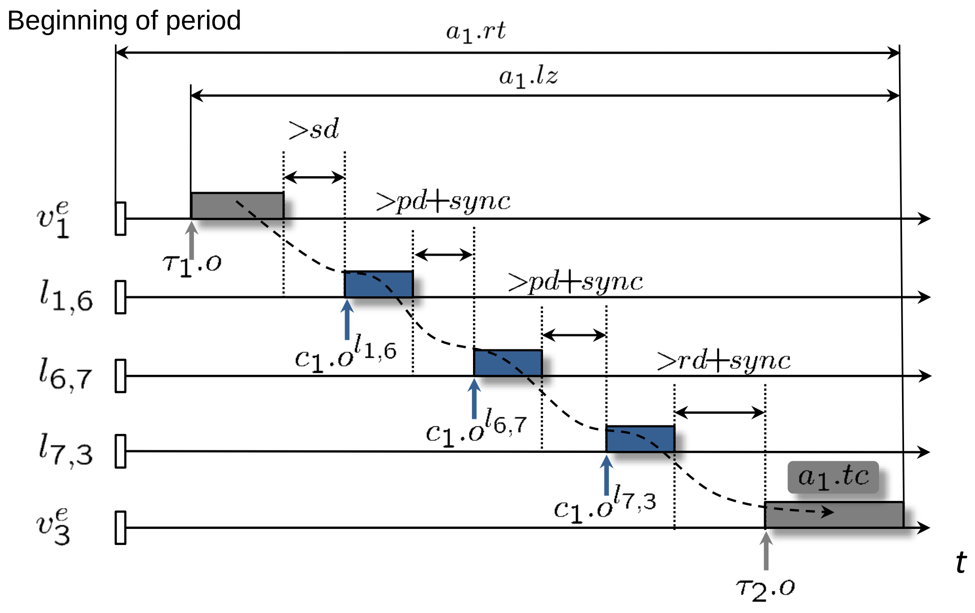

Figure 2.6: This figure shows the task chain and schedules of application a_1 in the example in Figure 2.4.

Three applications a_1, a_2 and a_3 are shown in Figure 2.4. We further denote the tasks contained in the task chain as $a_i.t_j$, where $1 \leq j \leq \beta_i$ with $a_i.t_1$ and $a_i.t_{\beta_i}$ denoting the first and last task respectively. Each task in the task chain either represents an application or a communication task. Figure 2.6 shows the details of task chain $a_1.tc$. There are three tasks in a_1, $\beta_1 = 3$ with $a_1.t_1 = \tau_1$, $a_1.t_2 = c_1$ and $a_1.t_3 = \tau_2$. $a_i.p$ denotes the period of the application and all the application and communication tasks share the same period with the application. That is, $\tau_1.p = c_1.p = \tau_2.p = a_1.p$ in the above example. Finally, $a_i.rt$, $a_i.lz$ are the *response time* and the *end-to-end latency*, which represent the time between the start of the period and the end of last task and the time between the start of the first task and end of last task. The significance of these two parameters will be discussed in details in the following sections.

Additional timing restrictions: In this work, we consider all the end stations as single processors running under a time-triggered non-preemptive scheduling scheme. Such scheduling scheme is common in safety-critical domains in particular. If an application task finishes its execution, it needs some time before the data can be packed in frames and sent on the network. This time interval has an upper bound which is denoted by *sd*. Similarly, once a frame arrives at an end station, it needs some time to get unpacked and processed before the data can be utilized by the corresponding application task. We assume that this time has an upper bound denoted as *rd*. On the network, we use the general case where each Ethernet switch possesses a *dispatcher* and each frame arriving at a switch is *forwarded* according to a schedule. The maximum processing delay of a frame in a switch is bounded by *pd*. That is, *pd* is the time between reception of the last bit on the *input port* and the earliest possible transmission of the first bit on the *output port.* It should be noted that *store-and-forward* and *cut-through* mechanism [13] can be modeled by setting *pd* an appropriate value. We further denote the *bandwidth* as *bw* – time to send one bit on the link and *interframe gap* as *ifg* – minimal link idle time between the transmission of two consecutive frames. The *precision*, i.e., the maximum difference between the any two clocks in the system is denoted as *sync*. In this work, all the schedules are referenced to the local time of the network nodes and we pad a *sync* in the constraints with schedules from different nodes.

2.4 Approach

In this work, we present a framework to co-synthesize schedules for all the application tasks τ_i and communication tasks c_i such that some high-level objectives are optimized. We characterize the distributed system described in the previous section by the following set of constraints and objectives.

2.4.1 Constraints

(C1) Collision free application tasks: This condition ensures that on every single processor, one application task is triggered only when the processor is idle, i.e., after the last task is finished. We define the set of all application tasks as $\mathcal{T}$ and that of those mapped on an end station v_m^e as $\mathcal{T}(v_m^e)$. This condition can be formulated as:

$$\begin{aligned}
&\forall m, v_m^e \in \mathcal{V}, \forall i, j, i \neq j, \tau_i, \tau_j \in \mathcal{T}(v_m^e) \\
&\tau_i.p \times k_i + \tau_i.o + \tau_i.e < \tau_j.p \times k_j + \tau_j.o \\
&\qquad\qquad\qquad\quad \textbf{or} \\
&\tau_j.p \times k_j + \tau_j.o + \tau_j.e < \tau_i.p \times k_i + \tau_i.o
\end{aligned} \tag{2.1}$$

where

$$\begin{aligned}
\forall k_i &\in \left[0, \frac{LCM(\tau_i.p, \tau_j.p)}{\tau_i.p} - 1\right] \\
\forall k_j &\in \left[0, \frac{LCM(\tau_i.p, \tau_j.p)}{\tau_j.p} - 1\right].
\end{aligned}$$

and $LCM(\tau_i.p, \tau_j.p)$ denotes the least common multiple of periods $\tau_i.p$ and $\tau_j.p$. In Figure 2.4, $\mathcal{T} = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5\}$ and $\mathcal{T}(v_1^e) = \tau_1$ for example. Although there is only one application task mapped on each end station in this example, there can be multiple applications tasks running on an end station as we will see in the experimental section. The constraint (2.1) is applicable to those scenarios.

(C2) Collision free communication tasks: To ensure there is no collision of frames being sent on a single directed Ethernet link, a frame can only start its transmission *ifg* time units after the last frame is finished. We define $\mathcal{C}$ as the set of all communication tasks and $\mathcal{C}(l_{m,n})$ as the set whose path trees contain link $l_{m,n}$. This condition can be formulated as:

$$\begin{gathered}
\forall m, n, l_{m,n} \in \mathcal{E}, \forall i, j, c_i, c_j \in \mathcal{C}(l_{m,n}) \\
c_i.p \times k_i + c_i.o^{l_{m,n}} + f_i.fl/bw + ifg < c_j.p \times k_j + c_j.o^{l_{m,n}} \\
\textbf{or} \\
c_j.p \times k_j + c_j.o^{l_{m,n}} + f_j.fl/bw + ifg < c_i.p \times k_i + c_i.o^{l_{m,n}}
\end{gathered} \tag{2.2}$$

where

$$\begin{aligned}
\forall k_i &\in \left[0, \frac{LCM(c_i.p, c_j.p)}{c_i.p} - 1\right] \\
\forall k_j &\in \left[0, \frac{LCM(c_i.p, c_j.p)}{c_j.p} - 1\right].
\end{aligned}$$

In Figure 2.4, $\mathcal{C} = \{c_1, c_2\}$, $\mathcal{C}(l_{6,8}) = \{c_1\}$ and similarly, $\mathcal{C}(l_{7,6}) = \{c_2\}$. Although there is only one communication task mapped on each link $l_{m,n}$ in this example, there can be multiple communication tasks sharing the same link. The constraint (2.2) is applicable to those scenarios.

(C3) Path dependency in communication: In a communication task, a frame can only be forwarded along the paths in the correct temporal order. We further represent the schedule $c_i.o^{l_{m,n}}$ on the link $l_{m,n}$ as $c_i.o^{l_{m,n}}[ph_j, q]$ or simply $c_i.o[ph_j, q]$, i.e., the q^{th} schedule in path ph_j. Here, $q = 1$ and $q = \gamma_{i,j}$ represent the first and last schedule. The data dependency of the communication can be formulated as:

$$\begin{gathered}
\forall i, c_i \in C, \forall j \in [1, \alpha_i], \forall q \in [2, \gamma_{i,j}] \\
c_i.o[ph_j, q-1] + f_i.fl/bw + pd + sync < c_i.o[ph_j, q] \ .
\end{gathered} \tag{2.3}$$

It should be noted that $f_i.fl/bw$ represents the transmission time of the frame f_i for a given bandwidth bw. In Figure 2.5, let us consider the path $c_1.ph_1$. Here, $\gamma_{1,1} = 3$ and the three schedules $c_1.o^{l_{1,6}}[ph_1, 1]$, $c_1.o^{l_{6,7}}[ph_1, 2]$ and $c_1.o^{l_{7,3}}[ph_1, 3]$ should be in the correct temporal order satisfying the constraint (2.3). Note that $c_1.o^{l_{1,6}}[ph_1, 1]$, $c_1.o^{l_{1,6}}[ph_2, 1]$ and $c_1.o^{l_{1,6}}[ph_3, 1]$ represent the same schedule since all three paths share the same link $l_{1,6}$.

(C4) Data dependency in applications: Due to data dependency, the application and communication tasks in the task chain of an application have to be executed in the correct temporal order. In a task chain, two consecutive tasks must be one of the following cases: (i) both application tasks, (ii) an application task followed by a communication task (iii) a communication task followed by an application task. We define the set of all applications as $\mathcal{A}$. This condition can be formulated as follows:

$$\begin{aligned}
&\forall i, a_i \in \mathcal{A}, \forall j \in [1, \beta_i - 1] \\
&\textbf{if} \;\; a_i.t_j = \tau_h \in \mathcal{T} \wedge a_i.t_{j+1} = \tau_g \in \mathcal{T} \;\textbf{then} \\
&\quad \tau_h.o + \tau_h.e < \tau_g.o && (2.4) \\
&\textbf{if} \;\; a_i.t_j = \tau_h \in \mathcal{T} \wedge a_i.t_{j+1} = c_g \in \mathcal{C} \;\textbf{then} \\
&\quad \tau_h.o + \tau_h.e + sd < c_g.o[ph_u, 1] && (2.5) \\
&\textbf{if} \;\; a_i.t_j = c_h \in \mathcal{C} \wedge a_i.t_{j+1} = \tau_g \in \mathcal{T} \;\textbf{then} \\
&\quad c_h.o[ph_v, \gamma_{h,v}] + f_h.fl/bw + sync + rd < \tau_g.o \;. && (2.6)
\end{aligned}$$

where ph_u is any path within the path tree and ph_v represents the path which is utilized by the corresponding application. For illustration, let us consider the example in Figure 2.4. For application a_1 in Figure 2.6, the application task τ_1 is followed by c_1 and the schedules $\tau_1.o$ and $c_1.o^{l_{1,6}}$ are constrained by (2.5). Similarly, c_1 is followed by τ_2 and the schedules $c_1.o^{l_{7,3}}$ and $\tau_2.o$ are constrained by (2.6).

(C5) Application response time constraint: The response time of an application a_i is given by:

$$a_i.rt = a_i.t_{\beta_i}.o + a_i.t_{\beta_i}.e(+sync) \;. \tag{2.7}$$

We denote the response time of an application as the time when the last task in the corresponding task chain is finished from the beginning of period. If observed from the local time of the sender station, the *sync* can be omitted. Figure 2.7 shows an example of response time of two tasks a_i and a_j. In a_i, $a_i.t_{\beta_1}.o \neq$ *beginning of period* possibly because of unavailability of (computation or network) resources. In many real-life time-triggered systems, the platform status information needs to be updated within a maximum tolerable time bound which is reflected by response time of an application. A hard constraint on maximum tolerable response time $a_i.rt_{max}$ can be enforced by adding the condition

$$a_i.rt < a_i.rt_{max}, \forall i, a_i \in \mathcal{A} \;. \tag{2.8}$$

(C6) Application end-to-end latency constraint: End-to-end latency of an application can be formulated as:

$$a_i.lz = a_i.t_{\beta_i}.o + a_i.t_{\beta_i}.e - a_i.t_1.o(+sync) \;. \tag{2.9}$$

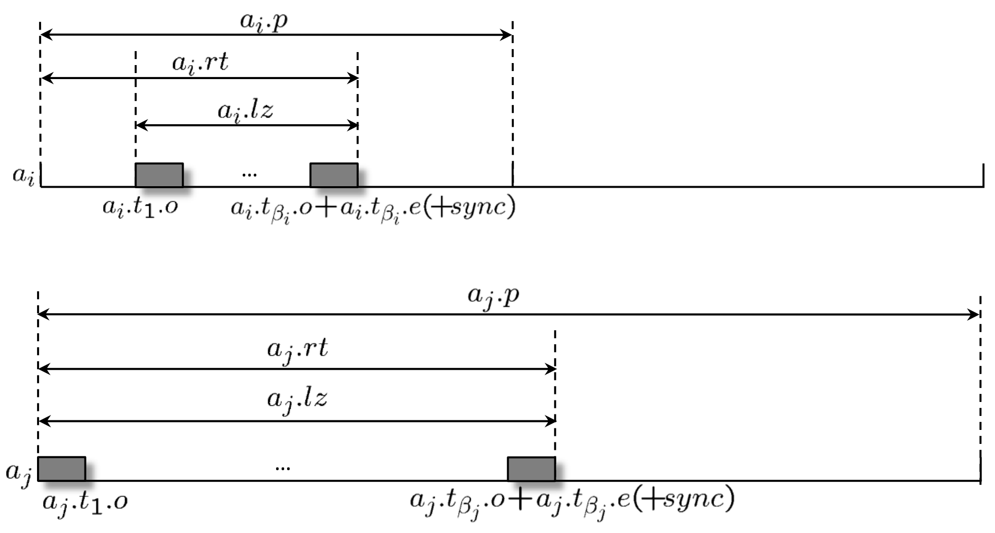
$a_i.p$
$a_i.rt$
$a_i.lz$
a_i
...
$a_i.t_1.o$
$a_i.t_{\beta_i}.o + a_i.t_{\beta_i}.e(+sync)$
$a_j.p$
$a_j.rt$
$a_j.lz$
a_j
...
$a_j.t_1.o$
$a_j.t_{\beta_j}.o + a_j.t_{\beta_j}.e(+sync)$

Figure 2.7: Response time and end-to-end latency.

Clearly, the end-to-end latency of an application is the time between the start of the first task and the finishing of the last task of its task chain. Similar to the response time, a maximum tolerable end-to-end latency can be enforced by the condition

$$a_i.lz < a_i.lz_{max}, \forall i, a_i \in \mathcal{A} \ . \tag{2.10}$$

Figure 2.7 illustrates the difference between the response time and the end-to-end latency. When $a_i.t_{\beta_1}.o =$ *beginning of period*, $a_i.rt = a_i.lz$. The application a_2 in Figure 2.7 shows such an example. On the other hand, when $a_i.t_{\beta_1}.o \neq$ *beginning of period*,

$$a_i.rt = a_i.lz + a_i.t_1.o \ .$$

One such example is the application a_i in Figure 2.7.

2.4.2 Constraint Formulation as MIP

Here, we discuss how the constraints and optimization objectives described in the previous section can be converted into a MIP problem. Towards this, each constraint should be represented as a single inequity or equation. The constraints (C3) to (C6) are already in the form of a single inequity. Since the constraints (C1) and (C2) are *either-or* conditions, they have to be converted by introducing a binary decision variable [95]. Thus, (C1) can be represented as

$$\begin{aligned}
&\forall n, v_n^e \in V, \forall i,j, i \neq j, \tau_i, \tau_j \in T(v_m^e), \forall q \in [1,\ \lambda] \\
&\tau_i.p \times k_i + \tau_i.o + \tau_i.e < \tau_j.p \times k_j + \tau_j.o + y_q \times M_q \\
&\tau_j.p \times k_j + \tau_j.o + \tau_j.e < \tau_i.p \times k_i + \tau_i.o + (1 - y_q) \times M_q.
\end{aligned} \tag{2.11}$$

where y_q denotes the introduced binary variable, λ represents the total number of possible collisions between application tasks and M_q is a sufficiently large constant. Similar conversion can be applied to (C2).

2.4.3 Objectives

Given the above representation of a time-triggered system, we consider three different classes of objectives. Let us consider a set of N applications $\mathcal{A}(obj) \in \mathcal{A}$.

(O1) Max/avg response time:

$$\begin{aligned}
&\forall i, a_i \in \mathcal{A}(obj) \\
&\mathcal{A}(obj).rt_{max} = max(a_i.rt) \\
&\mathcal{A}(obj).rt_{avg} = \sum a_i.rt/N
\end{aligned} \tag{2.12}$$

(O2) Max/avg end-to-end latency

$$\begin{aligned}
&\forall i, a_i \in \mathcal{A}(obj) \\
&\mathcal{A}(obj).lz_{max} = max(a_i.lz) \\
&\mathcal{A}(obj).lz_{avg} = \sum a_i.lz/N
\end{aligned} \tag{2.13}$$

(O3) Multi-objective optimization : Multi-objective optimization problem is also important. For example, a system designer may want to minimize the maximal response time of all the applications while putting certain emphasis on several applications with more stringent timing requirements. Here we formulate such multi-optimization problem by formulating the overall objective function as a weighted sum of all sub-objectives. The designer can adjust the weight according to the timing requirement of the system.

$$\begin{aligned}
&\forall i, obj_i \in \mathcal{OBJ} \\
&obj_M = \sum obj_i \times w_i
\end{aligned} \tag{2.14}$$

where $\mathcal{OBJ}$ denotes the set of objectives in the form of (O1) or (O2) and obj_M denotes the weighted multi-objective function.

2.4.4 Objective Formulation as MIP

The implementation of objective function as the response time or end-to-end latency of a single application or the average value of multiple applications is straight forward. When the objective is the maximum value of several applications, the *minimax* problem can be formulated in MIP by introducing a continuous variable z [95]. Thus the minimax objective functions can be converted to

$$obj = z, \forall i, a_i \in \mathcal{A}(obj), a_i.rt \leq z, a_i.lz \leq z \; . \tag{2.15}$$

Such an objective will introduce an extra continuous variable z and N inequities, which must be added to the constraints. Further, a multi-objective function can be formulated as (2.14).

2.4.5 Generalized Constraints and Objectives

In the aforementioned constraints, we have taken the following assumptions: (i) The execution of the whole application is finished within the period of the application and there is no application spanning across different periods. (ii) The release time of an application is at the beginning of the period. (iii) the response time and the end-to-end latency of an applications is smaller than the sampling period. These assumptions are valid for a system requiring ultra-low end-to-end delay and jitter, which is the purpose of implementing a such a time-triggered system. The constraints and the objectives can be relaxed for a more generalized case as the following

(C7) Generalized path dependency constraint: The path dependency constraint in (C3) can further be extended in to the following form, where $\sigma^c_{i,q}$ is an integer variable taking the value of 0 to 2. When σ takes the value 0, it is an identical constraint as (C3). However, this generalization allows the consideration of scenarios in the case of $\sigma = 1$ and $\sigma = 2$ in Figure 2.8.

$$\begin{aligned}&\forall i, c_i \in C, \forall j \in [1, \alpha_i], \forall q \in [2, \gamma_{i,j}]\\ &c_i.o[ph_j, q-1] + f_i.fl/bw + pd + sync < c_i.o[ph_j, q] + c_i.p \cdot \sigma^c_{i,q}.\end{aligned} \tag{2.16}$$

(C8) Generalized data dependency constraint: Similar to (C7), for the data dependency constraint, (C4) can be extended to the following form, where $\sigma^t_{i,j}$ again is an integer variable.

$$\forall i, a_i \in \mathcal{A}, \forall j \in [1, \beta_i - 1]$$

$$\textbf{if} \;\; a_i.t_j = \tau_h \in \mathcal{T} \wedge a_i.t_{j+1} = \tau_g \in \mathcal{T} \;\textbf{then}$$

$$\tau_h.o + \tau_h.e < \tau_g.o + a_i.p \cdot \sigma^t_{i,j} \tag{2.17}$$

$$\textbf{if} \;\; a_i.t_j = \tau_h \in \mathcal{T} \wedge a_i.t_{j+1} = c_g \in \mathcal{C} \;\textbf{then}$$

$$\tau_h.o + \tau_h.e + sd < c_g.o[ph_u, 1] + a_i.p \cdot \sigma^t_{i,j} \tag{2.18}$$

$$\textbf{if} \;\; a_i.t_j = c_h \in \mathcal{C} \wedge a_i.t_{j+1} = \tau_g \in \mathcal{T} \;\textbf{then}$$

$$c_h.o[ph_v, \gamma_{h,v}] + f_h.fl/bw + sync + rd < \tau_g.o + a_i.p \cdot \sigma^t_{i,j}. \tag{2.19}$$

Considering (C7) and (C8), we can also extend the definition of the end-to-end latency and the response time to the following form.

(D1) Generalized application end-to-end latency:

$$a_i.lz = a_i.t_{\beta_i}.o + a_i.t_{\beta_i}.e - a_i.t_1.o + +a_i.p \cdot (\sum_j \sigma^t_{i,j} + \sum_{k,q} \sigma^c_{k,q})(+sync). \tag{2.20}$$

(D2) Generalized application response time:

$$a_i.rt = a_i.t_{\beta_i}.o + a_i.t_{\beta_i}.e - a_i.rl + a_i.p \cdot (\sum_j \sigma^t_{i,j} + \sum_{k,q} \sigma^c_{k,q})(+sync). \tag{2.21}$$

where $\sigma^t_{i,j}$ and $\sigma^c_{k,q}$ are the integer variables introduced in (C7) and (C8). In addition, we further define a release time of the application that is not at the beginning of the period as $a_i.rl$ for the response time. In an optimization problem, when the values of (D1) and (D2) are under minimization, the solver will automatically choose the minimal possible value of $\sigma^t_{i,j}$ and $\sigma^c_{k,q}$. As already mentioned, the values of $\sigma^t_{i,j}$ and $\sigma^c_{k,q}$ can take 0, 1 or 2, corresponding to the three cases depicted in Figure 2.8. This extension allows the schedule co-synthesis problem to consider a more generalized case and also allows the end-to-end latency and response time larger than the period, cross-period applications. In the case where the timing constraints are smaller than the period, σ can be replaced by a binary decision variable, taking the value of 0 or 1.

2.5 Experimental Results

In this section, we present the experimental results to show the applicability of our approach. We present an industrial size case study of a distributed system to illustrate the various design aspects considering different optimization objectives. Further, we present a

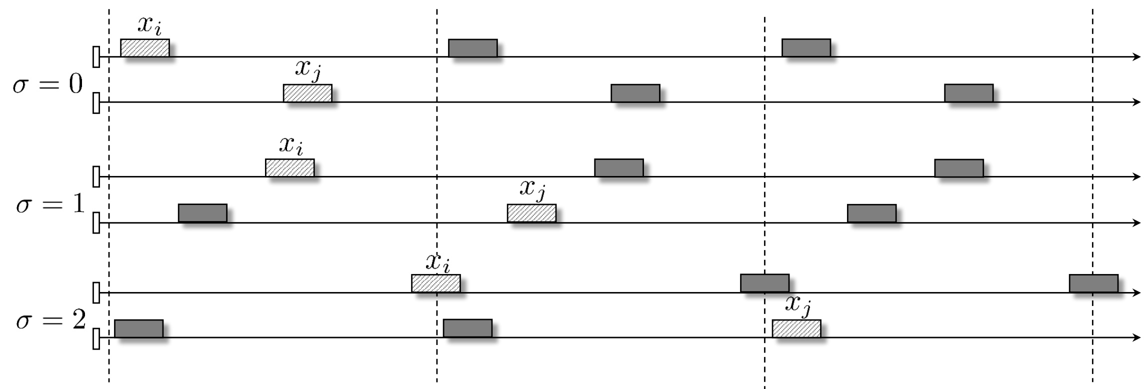

Figure 2.8: This figure shows the possible values of σ and the corresponding scenario. Here we abstract the task execution and the frame transmission to just a process x_i and x_j.

scalability analysis and show that our formulation scales to systems with reasonably large size. We used Gurobi 5.10 [88] for solving the MIP model and all experiments are carried out on a computer with 1.87GHz CPU and 4GB memory.

2.5.1 Case Study

In the case study, we consider a distributed system consisting of 12 end stations connected via a switched Ethernet network. We explore four different network topologies as shown in Figure 2.9. For the ring topology, the frames are only allowed to be forwarded in one direction in the ring. Further, 53 application tasks are mapped on the processors and 23 frames are sent on the network amongst which 7 are multi-cast. The application tasks and communication tasks constitute a total of 30 applications, with 5 of them having a task chain length of 5. The system parameters are bw = 100 Mbps, ifg = 0.96 us (12 byte), $sd = pd = rd$ = 10 us, $sync$ = 5 us. The details of the system configuration is shown in Table 2.1 to Table 2.3. The schedule synthesis problem is formulated with constraints (C1) to (C6) and objective (O1) to (O3).

In the experiment, we define the following objectives to be minimized (response time and latency are observed from local time of the first sender station):

- obj_1: Maximal response time of all applications $\mathcal{A}$.
- obj_2: Maximal response time of applications a_1 to a_5.
- obj_3: Maximal response time of applications a_1 to a_{10}.
- obj_4: Average response time of all applications $\mathcal{A}$.
- obj_5: Maximal end-to-end latency of all applications $\mathcal{A}$.

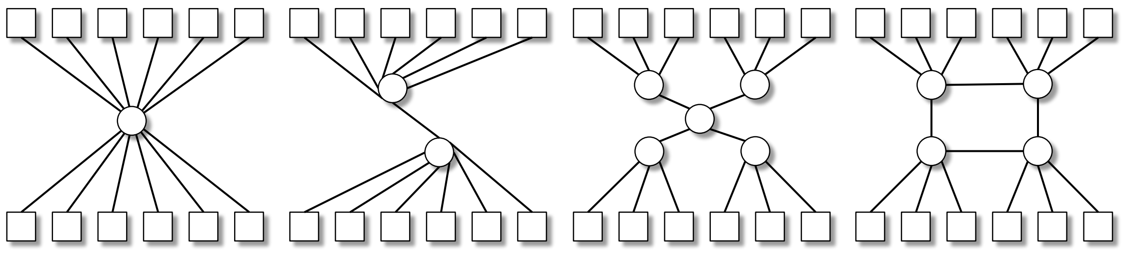

Figure 2.9: Network topologies explored (from left to right): (a) Star, (b) Twin-star, (c) Tree, (d) Ring.

v_i^e	τ_i	$\tau_i.e$[us]	send c_i	receive c_i
v_1^e	τ_1,τ_2,τ_3,τ_4,τ_5	200	c_1,c_2,-,-,-	-,-,c_3,c_5,c_7
v_2^e	τ_6,τ_7,τ_8,τ_9,τ_{10}	500	c_3,c_4,-,-,-	-,-,c_{10},c_{17},c_{21}
v_3^e	τ_{11},τ_{12},τ_{13},τ_{14}	550	c_5,c_6,-,-	-,-,c_2,c_4
v_4^e	τ_{15},τ_{16}	350	c_7,c_8	c_1,c_{13}
v_5^e	τ_{17},τ_{18},τ_{19},τ_{20},τ_{21}	500	c_9,-,-,-,-	-,c_6,c_{11},c_{17},c_{18}
v_6^e	τ_{22},τ_{23},τ_{24},τ_{25},τ_{26},τ_{27}	400	c_{10},c_{11},c_{12},-,-,-	-,-,-,c_{14},c_{15},c_{18}
v_7^e	τ_{28},τ_{29},τ_{30},τ_{31},τ_{32}	300	c_{13},c_{14},-,-,-	-,-,c_8,c_{12},c_{19}
v_8^e	τ_{33},τ_{34},τ_{35},τ_{36},τ_{37}	500	c_{15},c_{16},-,-,-	-,-,c_9,c_{23},c_3
v_9^e	τ_{38},τ_{39},τ_{40},τ_{41}	500	c_{17},-,-,-	-,c_{16},c_{21},c_9
v_{10}^e	τ_{42},τ_{43},τ_{44},τ_{45}	300	c_{18},c_{19},-,-	-,-,c_{17},c_{21}
v_{11}^e	τ_{46},τ_{47},τ_{48},τ_{49},τ_{50}	500	c_{20},-,-,-,-	-,c_3,c_{19},c_{22},c_{18}
v_{12}^e	τ_{51},τ_{52},τ_{53}	600	c_{21},c_{22},c_{23}	c_3,c_{10},c_{20}

Table 2.1: Configuration of application tasks.

Depending on the higher-level goals one or more of the objectives above can be optimized. For platform/system status applications, the response time is an important parameter since they need to be completed as early as possible. Towards this, obj_1 and obj_4 are useful objectives. Often, a subset of applications are system status applications while the rest are usual applications. In such cases, obj_2 and obj_3 allow to minimize the response time of the relevant applications. Moreover, for many applications such as feedback control loops, the maximum end-to-end latency plays an important role. Therefore, obj_5 also has application-level relevance. In general, it is possible to form various combinations of response time and end-to-end latency as an objective depending on the exact design requirements. Further, in many real-life scenarios, a single objective might not suffice. In such cases, our approach offers a multi-objective formulation. In the case study, we explore the results considering each single objective and several multi-objective cases. In the case of multi-objective optimization, we assign equal weights w_i.

a_i	$a_i.p$[ms]	$a_i.tc$	a_i	$a_i.p$[ms]	$a_i.tc$
a_1	5	τ_1,c_1,τ_{15},c_7,τ_5	a_{16}	4	τ_{28},c_{13},τ_{16},c_8,τ_{30}
a_2	10	τ_2,c_2,τ_{13}	a_{17}	4	τ_{29},c_{14},τ_{25}
a_3	5	τ_6,c_3,τ_3	a_{18}	20	τ_{33},c_{15},τ_{26}
a_4	5	τ_6,c_3,τ_{37}	a_{19}	20	τ_{34},c_{16},τ_{39}
a_5	5	τ_6,c_3,τ_{47}	a_{20}	10	τ_{38},c_{17},τ_9
a_6	5	τ_6,c_3,τ_{51},c_{21},τ_{45}	a_{21}	10	τ_{38},c_{17},τ_{20}
a_7	10	τ_7,c_4,τ_{14}	a_{22}	10	τ_{38},c_{17},τ_{44}
a_8	10	τ_{11},c_5,τ_4	a_{23}	10	τ_{42},c_{18},τ_{21}
a_9	10	τ_{12},c_6,τ_{18}	a_{24}	10	τ_{42},c_{18},τ_{27}
a_{10}	10	τ_{17},c_9,τ_{35}	a_{25}	10	τ_{42},c_{18},τ_{50}
a_{11}	10	τ_{17},c_9,τ_{41}	a_{26}	10	τ_{43},c_{19},τ_{32}
a_{12}	5	τ_{22},c_{10},τ_8	a_{27}	10	τ_{43},c_{19},τ_{48}
a_{13}	5	τ_{22},c_{10},τ_{52},c_{22},τ_{49}	a_{28}	5	τ_{46},c_{20},τ_{53},c_{23},τ_{36}
a_{14}	10	τ_{23},c_{11},τ_{19}	a_{29}	5	τ_{51},c_{21},τ_{10}
a_{15}	10	τ_{24},c_{12},τ_{31}	a_{30}	5	τ_{51},c_{21},τ_{40}

Table 2.2: Configuration of the applications.

c_i	$f_i.fl$[B]	c_i	$f_i.fl$[B]	c_i	$f_i.fl$[B]	c_i	$f_i.fl$[B]
c_1	64	c_7	100	c_{13}	100	c_{19}	100
c_2	80	c_8	100	c_{14}	150	c_{20}	64
c_3	64	c_9	64	c_{15}	150	c_{21}	64
c_4	80	c_{10}	64	c_{16}	100	c_{22}	64
c_5	80	c_{11}	64	c_{17}	100	c_{23}	64
c_6	100	c_{12}	64	c_{18}	100		

Table 2.3: Frame length of communication tasks.

2.5.2 Results and Discussions

The optimization results for the star and tree topology are shown in in Table 2.4 and for the twinstar and ring topology are shown in Table 2.5. Firstly, in the case of single objective optimization, the first five rows of Table 2.4 and Table 2.5 show the results for four different topologies. The results for the tree topology is further illustrated using Figure 2.10. Clearly, the optimality was achieved for each individual objective (in bold font). For example, as shown in Figure 2.10, the result for obj_1 for tree topology is $obj_1 = 2880.96$ us, which is the minimal value for obj_1 among all the cases of single-objective optimization. Similar results also can be drawn for obj_2 to obj_5 in Figure 2.10. Naturally, such single objective cases often lead to non-optimal results for the others. For example, in the case of tree topology, single objective optimization according to obj_2 and obj_3 leads to the minimal value of $obj_2 = 1342.48$ us and $obj_3 = 2200$ us respectively, but results in the undesirable results for obj_1 as 12395 us and 18995 us respectively. It should be noted that the results of the objectives that are not optimized (not underlined in Table 2.4 and Table 2.5) can vary

obj	STAR				
	1 [us]	2 [us]	3 [us]	4 [us]	5 [us]
1	**2800.48**	2800.48	2800.48	2164.52	2800.48
2	18995.00	**1256.24**	9995.00	5445.84	8944.76
3	12800.00	2200.00	**2200.00**	4601.62	6795.00
4	3006.48	**1256.24**	2206.00	**1590.88**	2550.24
5	10236.48	7785.76	7785.76	4485.34	**1700.48**
1+2	3006.48	**1256.24**	3006.48	2157.80	3006.48
1+3	**2800.48**	2200.00	**2200.00**	2141.23	2800.48
1+2+3	3006.48	**1256.24**	2206.00	1996.64	3006.48
1+4	**2800.48**	1650.24	2602.80	1630.01	2800.48
1+5	2900.48	2900.48	2900.48	2037.11	**1700.48**
1+2+3+4	3006.48	**1256.24**	2206.00	**1590.88**	2550.24
1+2+3+4+5	3056.48	1356.00	2206.00	1656.46	**1700.48**
obj	TREE				
	1 [us]	2 [us]	3 [us]	4 [us]	5 [us]
1	**2880.96**	2880.96	2880.96	1954.47	2880.96
2	12395.00	**1342.48**	9995.00	6087.15	9995.00
3	18995.00	2200.00	**2200.00**	5104.39	10601.48
4	3092.72	**1342.48**	2252.00	**1615.28**	2590.48
5	12962.20	8624.68	8624.68	6010.64	**1740.72**
1+2	3092.72	**1342.48**	2902.00	2062.89	3092.72
1+3	**2880.96**	2200.00	**2200.00**	2027.25	2880.96
1+2+3	3092.72	**1342.48**	2252.00	2077.51	3092.72
1+4	**2880.96**	1690.48	2602.80	1659.99	2880.96
1+5	2902.00	2703.28	2902.00	2021.08	**1740.72**
1+2+3+4	3092.72	**1342.48**	2252.00	**1615.28**	2487.68
1+2+3+4+5	3132.96	1356.00	2252.00	1684.95	1750.00

Table 2.4: Results for the star and tree topology according to different optimization objectives.

under different solver configurations. This is because it is possible that there are multiple solutions to such an optimization problem and the results in the table show only one of them. However, the results of objectives that are considered for optimization are independent of the solver configuration.

Therefore, in order to achieve more balanced optimization results, multi-objective optimization is more desirable. In the case of multi-objective optimization, it is often possible to achieve results close to the optimal one for all the objectives under consideration. We again illustrate the optimization results using the tree topology. We first consider an example shown in Figure 2.11. If the synthesis is optimized towards a combination of obj_1, obj_2 and obj_3, the results obtained are $obj_1 = 3092.72$ us, $obj_2 = 1342.48$ us, $obj_3 = 2252$ us. Clearly, all three objectives are close or equal to the results in the single objective cases. Similar results can be observed for the case of obj_1, obj_4 and obj_5, as shown in Figure 2.12.

obj	TWINSTAR				
	1 [us]	2 [us]	3 [us]	4 [us]	5 [us]
1	**2820.60**	2820.60	2820.60	1920.03	2820.60
2	11126.36	**1256.24**	9995.00	5147.35	6027.00
3	11200.00	1650.00	**2200.00**	4600.77	8188.52
4	3006.48	1570.36	2206.00	**1597.34**	2570.36
5	12504.96	9166.24	9166.24	6295.11	**1700.48**
1+2	3006.48	**1256.24**	2906.48	1801.53	3006.48.
1+3	**2820.60**	2200.00	**2200.00**	1979.30	2820.60
1+2+3	3006.48	**1256.24**	2260.00	2009.12	3006.48
1+4	**2820.60**	1670.36	2602.80	1638.19	2820.60
1+5	2900.48	2900.48	2900.48	1961.06	**1700.48**
1+2+3+4	3006.48	**1256.24**	2206.00	**1597.34**	2467.56
1+2+3+4+5	3056.48	1356.00	2206.00	1689.81	**1700.48**

obj	RING				
	1 [us]	2 [us]	3 [us]	4 [us]	5 [us]
1	**2840.72**	2840.72	2840.72	2093.28	2840.72
2	12395.00	**1299.36**	9995.00	5915.79	8186.04
3	18995.00	2200.00	**2200.00**	5569.66	11904.52
4	3049.60	**1299.36**	2229.00	**1604.08**	2570.36
5	14464.28	6170.60	6170.60	4703.60	**1720.60**
1+2	3049.60	**1299.36**	3049.60	2290.29	3049.60
1+3	**2840.72**	2200.00	**2200.00**	2009.24	2840.72
1+2+3	3049.60	**1299.36**	2229.00	1915.68	3049.60
1+4	**2840.72**	1670.36	2602.80	1651.58	2840.72
1+5	2900.48	2683.16	2900.48	1972.93	**1720.60**
1+2+3+4	3049.60	**1299.36**	2229.00	**1604.08**	2570.36
1+2+3+4+5	3076.60	1256.00	2229.00	1682.22	**1720.60**

Table 2.5: Results for the twinstar and ring topology according to different optimization objectives.

Further, we investigate the influence of the weights on the optimization results. Figure 2.13 shows the result in the case of multi-objective optimization of obj_1 and obj_4 for different weight ratios w_4/w_1. As expected, the weight ratio of the objective can influence the optimization result. A higher w_4/w_1 should imply a relatively lower obj_4 which is also reflected in Figure 2.13.

From the experimental results we can see that our formulation allows the designer to generate schedules according to various application-level objectives. It also offers the possibility to combine single objectives to form multi-objective optimization problems that could meet more complicated higher-level requirements. Moreover, our approach is applicable to any network topology and configuration of applications, application tasks and communication tasks.

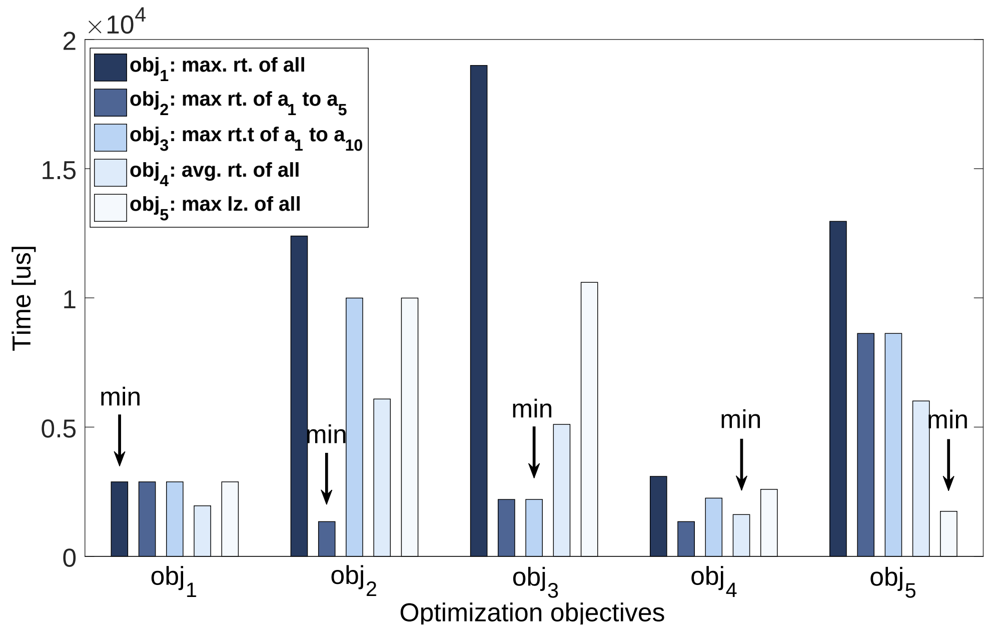

Figure 2.10: This figure shows the results for the single-objective optimization in the tree topology. Detailed data are documented in Table 2.4.

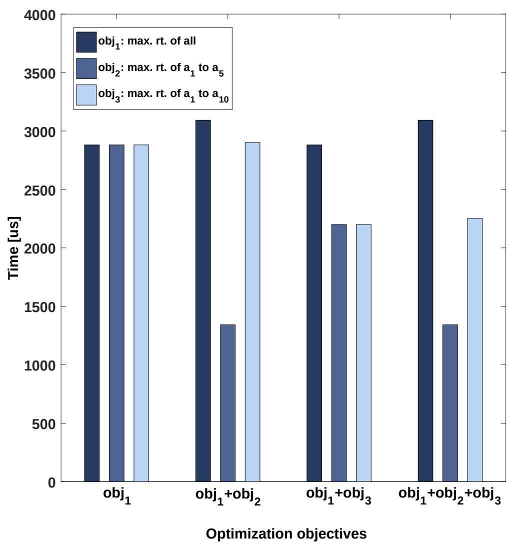

Figure 2.11: This figure shows the results for the multi-objective optimization in the tree topology considering obj_1, obj_2 and obj_3. Detailed data are documented in Table 2.4.

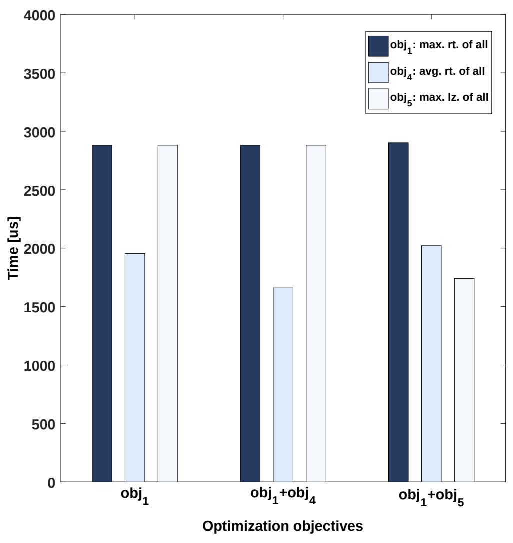

Figure 2.12: This figure shows the results for the multi-objective optimization in the tree topology considering obj_1, obj_4 and obj_5. Detailed data are documented in Table 2.4.

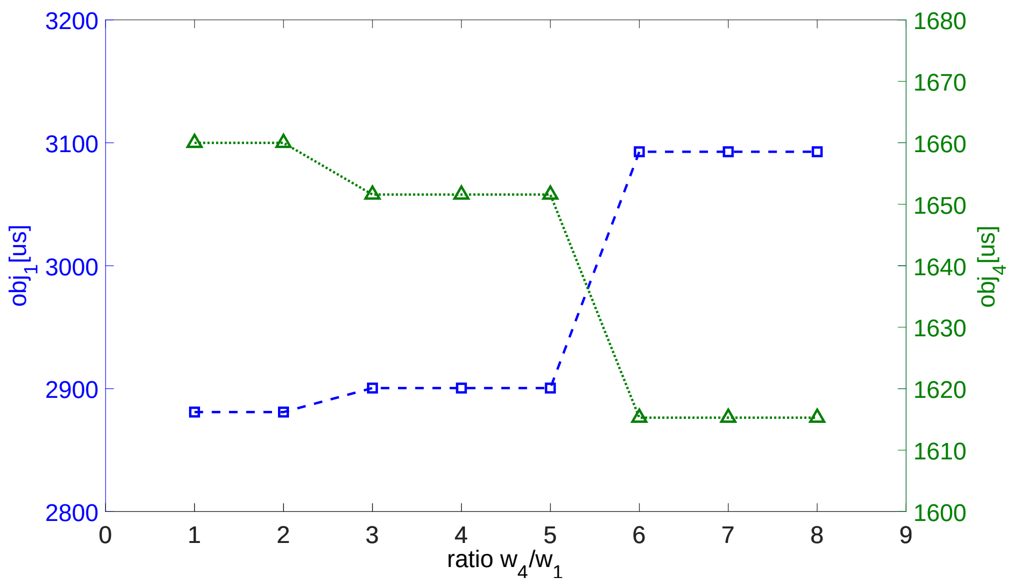

Figure 2.13: Results for obj_1 and obj_4 in a multi-objective optimization with different weight ratio w_4/w_1.

2.5.3 Scalability Analysis

To show the scalability of the proposed approach, we use a series of synthetic test systems in increasing size and measure the time required to solve the model. We construct 20 synthetic systems with different configurations for each system size from 9 to 90 applications. Initially, the systems employ a tree topology with three end stations and two switches. Subsequently, we add new applications and for each additional 9 applications, we add three end stations and one switch. All the applications consist of two application tasks and one uni-cast communication task. The periods of applications $a_i.p$ are chosen randomly from 4, 5, 10 and 20 ms and the WCET $\tau_i.e$ are between 100 and 300 us. Further, the frame lengths range from 80 to 200 bytes. Figure 2.14 shows the maximal, minimal and average runtime of the 20 synthetic cases. It can be observed from the figure that the average synthesis time for 90 applications is less than 200 seconds. Clearly, the proposed approach scales up to a system of this size, which is a reasonable size for many application domains such as an automotive ECU cluster. It is also interesting to note that the synthesis time for different systems of the same size varies a lot. For example, for a system consisting 90 applications, the case requiring shortest synthesis time needs about 20 seconds to synthesize where the case requiring longest time needs approximately 600 seconds. The reason for this variation lies within the system configuration and would be an interesting topic to explore for future work.

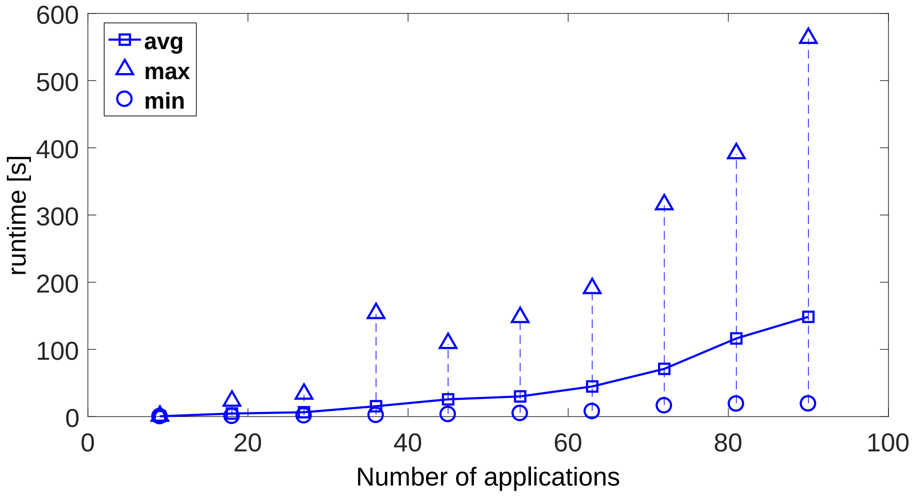

Figure 2.14: Number of applications vs. runtime.

2.6 Concluding Remarks

In this chapter, an approach to co-synthesize task and network schedules for an Ethernet-based time-triggered system is presented. We target at the time-triggered traffic in Ethernet and synthesize synchronous task and network schedules to optimize application-level timing objectives. The mathematical models of the constraints and the objectives in such a system is provided and the problem is formulated into a MIP problem. We further introduce the formulations for more generalized constraints. The proposed approach is independent of task and communication configurations as well as network topologies and device performance. An industrial-sized case study is used to show the applicability of the method towards optimization of complex timing objectives and the experimental results also show that it can be scaled to system of reasonably large size.

Real-time requirement is the main focus of the approach proposed in this chapter. As future work, this approach can be extended to the synthesis of task and network schedules according to further objectives like the extensibility of the schedules. Another possible future work in this direction is the examination of the relationship between the synthesis time and the system configuration. It has been shown in the scalability analysis in the experimental results that for the system of the same size, the required synthesis time varies a lot. The investigation of this problem could possibly lead to the identification of the relationship between system configuration and the computational effort of the synthesis. The results can certainly contribute to the scalability of similar synthesis approaches.

Chapter 3

Schedule Management for Cloud-based Automotive Software Systems

A framework for the schedule management problem for cloud-based future automotive software systems is presented in this chapter. The innovation in the automotive domain is shifting considerably to the E/E system and software. The evolution cycle of the electronic system and the software is significantly shorter than the life cycle of a vehicle and therefore the functionality of a vehicle might become 'outdated' easily in the future. Thus, it would be advantageous if new applications can be installed or existing applications can be upgraded via cloud services after sales in a PnP fashion. This requires that the underlying system possesses a certain degree of adaptivity and reconfigurability. One important issue in this case is the allocation of computation and communication resources. In the case of a time-triggered system, the task and network schedules need to be adapted to accommodate new applications. Towards addressing this problem, we propose a schedule management framework to obtain, synthesize and manage schedules efficiently online for Ethernet-based time-triggered systems in the automotive context. This framework is based on a client-server architecture and each side consists of a web module, a synthesis module and a configuration pool. It utilizes the Internet access of modern vehicles to exploit the computation and storage capacity on the server in a cloud-computing manner and can facilitate the reuse of generated schedule sets. In the synthesis module, a four-stage strategy is introduced to reduce the synthesis time and the disturbance to existing applications. The experimental results show that the proposed framework can be applied to generate and manage schedules online and benefit from both onboard and cloud-based schedule synthesis. The result also shows the applicability of the introduced four-stage synthesis strategy.

Chapter outline: This chapter is divided into six sections. Section 3.1 provides the introduction to the problem and explains the contributions. Section 3.2 then reviews the related work. The architectural setting and the formulation of the schedule management problem is explained in Section 3.3. This is followed by the proposed approach, which is presented in Section 3.4. The experimental results and evaluation are then provided in Section 3.5, which is followed by the concluding remarks in Section 3.6.

3.1 Introduction

Increasingly more software applications are being deployed in the automotive embedded systems in recent years, including, e.g., applications for driver assistance systems, infotainment and safety-critical control systems. These software applications might need constant updates from the OEM. Some OEMs are already offering after-market updates of certain softwares to the end user. Currently these software updates are limited to the infotainment domain, however, it is likely that in the future increasingly more software modules can be updated after sales. Moreover, novel applications can be designed and developed in a relatively shorter time, compared to the life cycle of a car. Therefore, in the future, it would be advantageous both for the car manufacturers and the customers if it is possible to install new applications that are developed after sales. Towards addressing this problem, one emerging trend in automotive embedded systems is the PnP capability of software applications, which enables the end user to install a new application or upgrade an existing one either by downloading it from the cloud or with a storage device like a USB stick. In a more futuristic scenario, which is in line with the autonomous vehicle concept, a vehicle can automatically detect the driving condition and download on demand the software applications that are suitable for the specific situation in a PnP fashion. For example, in heavy rain condition, a more sophisticated image processing algorithm can be downloaded and activated for the autonomous parking. In such a case, instead of pre-installing everything onboard the vehicle, the most up-to-date software functionalities can be accessed through cloud service when needed.

However, there still remain many challenges to be addressed for the PnP capability of applications. This is because the feature demands certain flexibility in the underlying system, in which case the application software, the system software and the communication services can be (partially) reconfigured to accommodate the new applications or new updates. However, current automotive embedded systems do not support PnP in a straight forward manner. One challenge is that according to the current automotive supply chain, ECUs are often equipped with one single independent function or an aggregate of several functions and the ECUs and their softwares are delivered by the supplier as black-boxes. However, as the size and complexity of the E/E architecture of vehicles increase, this might

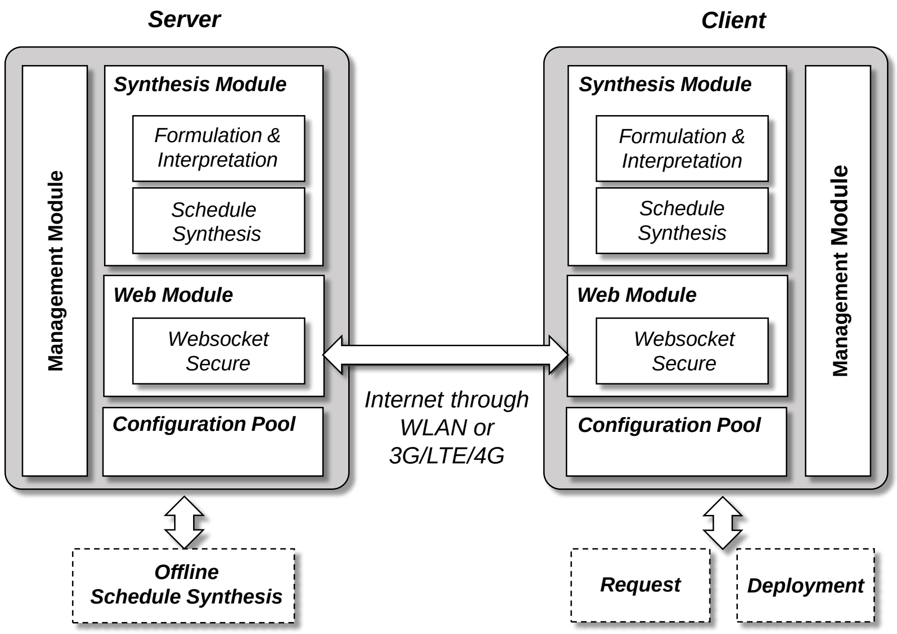

Figure 3.1: Software architecture of the proposed framework.

not be sustainable due to cost, space and complexity reasons. It is expected, that in the future, the E/E architecture of vehicles will shift from a federated architecture to an integrated architecture [40], where an ECU is treated as a computing platform and tasks from multiple applications can be mapped onto one ECU, as opposed to the current architecture, where each ECU is relatively function-specific. A further challenge lies in the deployment of new software components. Currently, the software on the ECU is built into one single binary file and flashed onto the hardware. Towards overcoming this hurdle, there are already some research works on operating systems and runtime environments that can accommodate new software modules [24, 96, 97] without re-flashing the whole ECU. In addition, there exist also proposals to extend AUTOSAR to allow the replacement of a software module [98].

Another important issue in a PnP scenario is the (re-)allocation of the computation and communication resources to accommodate the new applications or the updated applications with modified requirements on computation and communication resources. The work at hand addresses this problem, in particular, the schedule generation and management for an Ethernet-based time-triggered system in such context. Time-triggered paradigm is a typical design choice for an embedded system of stringent timing requirements, e.g., safety-critical control applications and other time-critical real-time applications, since it can deliver deterministic timing behavior. These time-triggered systems rely mainly on a schedule table, where the task triggering time and the transmission time of the messages is specified. In this chapter, we consider an Ethernet-based time-triggered distributed platform, which has been gaining ground in the industrial domains. Ethernet is considered one of the promising candidates for future in-vehicle communication network and is gradually replacing conventional automotive bus systems. Some more recent Ethernet protocols also support the time-triggered traffic. Examples of these protocols include the AS6802 [21] protocol and the upcoming TSN standards [15]. In such a system, the PnP feature imposes challenges on the resource re-allocation in several aspects, including, e.g., the generation of the schedules, the deployment of the schedules and the safety of the reconfiguration process. In this work, we focus on the problem of how the task and message schedules can be obtained, taking into account, the requirements for an after-market reconfiguration, e.g., a PnP scenario.

Typically such a schedule set is synthesized offline. However, to enable the PnP feature, a schedule set needs to be obtained online, so that the plug-in applications can be accommodated without taking the car to the manufacturer's garage. Modern premium vehicles are already equipped with the access to the Internet, and there are intensive research works on the connected car concept for allowing a vehicle to be able to connect to other vehicles or infrastructure units and servers. This connectivity can be utilized to provide solutions to this problem. Towards addressing this problem, several alternatives can be considered, including (i) pre-calculate the schedules offline, store them on a server and fetch them online, (ii) generate schedule sets onboard the vehicle, or (iii) utilizing the cloud-computing concept to generate schedule set on the server. Each of these alternatives has its advantages and disadvantages, which will be analyzed in Section 3.3.

In this chapter, we propose a schedule management framework for obtaining the schedules online for reconfiguration, combining the aforementioned alternatives. This software framework is based on a client-server architecture and contains on each side a configuration pool, a synthesis module and a web module. The configuration pool can store valid schedule configurations to facilitate their efficient reuse. In addition, schedule sets can also be pre-calculated offline and stored in the server to reduce the online synthesis effort. The synthesis module on both the client and the server can be used to calculate schedules. Here a four-stage re-scheduling strategy is introduced to offer a trade-off between the chance of accommodating new applications on one hand and the synthesis time and the disturbance to existing applications on the other. It includes a simple incremental design, two levels of conflict-based re-scheduling and a complete re-scheduling. The conflict-based re-scheduling is based on identifying the conflict of tasks and shared resources between the new and existing applications and re-schedules only existing applications with conflict. The web module, based on websocket secure, provides a secure communication channel between the client and the server, so that the client can fetch schedules from the server or use the computation capacity of the server for online schedule synthesis. This framework offers a balance between offline and online schedule synthesis and utilizes both the storage and computation capacity on the server to efficiently obtain and manage schedules in a reconfiguration scenario, e.g., installing new applications. The proposed software framework is implemented using a Raspberry Pi 2 as client to represent an onboard ECU and a laptop computer to represent a server, which can be managed and maintained by the car manufacturers. The result has shown the applicability of the proposed framework.

Contributions: In this chapter, we propose a schedule management framework that addresses the problem of generating and managing schedules for time-triggered systems in the context of PnP of future cloud-based automotive software systems. The framework utilizes the computation and storage capacity both onboard the vehicle and a server through cloud-computing. It can generate schedule sets online as well as store them and enable their reuse. We also introduce a four-stage scheduling scheme that starts from a simple incremental design, and step-by-step increases the number of existing applications to be re-scheduled based on identifying task and resource conflict. It offers a trade-off between the chance of accommodating new applications and the synthesis time as well as the disturbance to existing applications.

3.2 Related Works

The related research works can be roughly divided into four different categories.

(i) The schedule synthesis problem for time-triggered systems is a heavily researched subject. [59] has proposed an SMT formulation for the schedule synthesis problem for the time-triggered Ethernet traffic. [89, 94] have considered in addition how the time-triggered schedules can be synthesized taking the rate-constrained traffic also into consideration. [99, 100] addressed the problem of task and communication schedule co-synthesis. However, these approaches only addressed the offline schedule synthesis problem. On the other hand, this work at hand does not focus on proposing a new formulation, but addresses the problem of providing software framework support for the online schedule generation and management.

(ii) Several related works addressed the problem of incremental scheduling scheme. [71] has proposed heuristics based incremental design for a system based on TDMA protocol, with the objective to minimize the modification cost of existing schedules. [89] has considered the incremental design for time-triggered Ethernet using backtracking, but without any specific metric. The conflict-based incremental design method we propose in this framework has similar motivation as [71], but it can be fitted into a schedule synthesis framework without using additional optimization metrics.

(iii) There are also several works on the configuration and reconfiguration of time-triggered Ethernet networks. [101] has proposed a configuration agent which can be used to calculate new schedules based on traffic parameters learned. In addition, in [102], a method is proposed to analyze traffic parameters based on measurements. These related works mainly focused on the analysis of parameters for asynchronous traffic to make it synchronous, and not on a software framework addressing the challenges and requirements of obtaining schedules online, which is the focus of this work at hand.

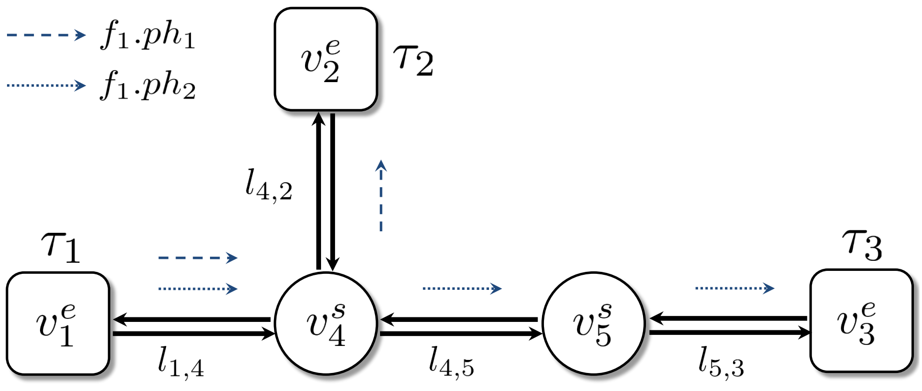

Figure 3.2: An example of Ethernet-based time-triggered system.

(iv) A few papers addressed the problem of PnP in the automotive setting. [96, 103] have considered the integration of new software components into the system. In [96], an Android-based framework is proposed that allows the plug-and-play of software components. [97, 104] proposed a runtime environment, which supports the plug-and-play of software components. However, [97, 104] did not address the problem of obtaining the schedule parameters for reconfiguration. [96] considered an online schedulablility analysis module for the priority-based scheduling scheme. But this cannot be applied to time-triggered systems, which require a schedule table.

3.3 Problem Formulation

3.3.1 Architectural Setting

Typically, an E/E architecture of a vehicle consists of several clusters of ECUs. Each cluster of ECUs are connected by a bus system, e.g., FlexRay, CAN or Ethernet. A distributed application is partitioned into a set of tasks. The tasks are often mapped on different ECUs and the data between the tasks are transmitted via messages over the communication bus. Some of the automotive applications have stringent timing requirements. Safety-critical applications are typical examples of such applications. Other non-safety-critical applications might also exhibit periodic nature and low latency and low jitter can considerably increase their performance. The time-triggered architecture has gained increasing popularity in recent years for its deterministic nature and the possibility of achieving low latency and jitter. Therefore, it is suitable for the aforementioned applications. In a time-triggered distributed system, tasks are executed and messages are transmitted according to pre-calculated schedules. Consequently the schedule calculation forms an important step in the design and development of such systems.

Ethernet-based Time-Triggered Distributed System

We consider an Ethernet-based time-triggered distributed platform, consisting of a set of ECUs communicating over a switched Ethernet network. In this setting, we consider the time-triggered non-preemptive scheduling scheme on the ECUs and the time-triggered traffic in the full-duplex switched Ethernet for communication. Furthermore, we consider that the clocks of the ECUs are synchronized to the network. A more detailed description of this architectural setting can be found in Section 2.3 of Chapter 2. Here we provide a brief recap of the system and adapt the notations slightly for the purpose of this chapter. The network architecture can be denoted as a graph $G(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ denote respectively the nodes (ECUs and switches) and the links between them. An ECU in the network can be denoted as v_i^e and a switch as v_i^s. A full-duplex Ethernet *link* is represented as $l_{m,n}$ and $l_{n,m}$, denoting respectively the directed link from nodes v_m to v_n and from v_n to v_m. Such a distributed system typically consists of a set of applications. Each *application* a_i is composed of a number of dependent tasks and the communication via network between the tasks. Each *task*, τ_i, is represented by a tuple $\{\tau_i.E, \tau_i.p, \tau_i.e, \tau_i.o\}$ denoting task mapping, period, offset and WCET respectively. Data between dependent tasks mapped on different ECUs are packed into frames and transferred over the Ethernet network. Each *frame* (similar to communication task in Chapter 2), f_i, is represented by a tuple $\{f_i.l, f_i.p, f_i.\mathbf{ph}, f_i.\mathbf{o}\}$ denoting the corresponding frame length, period, the set of dataflow paths and the set of link offsets respectively. Here, a *dataflow path* defines an ordered set of links from the source ECU to one destination ECU. In the case of multi-cast, a frame has several dataflow paths, each leading to one of the destination ECUs. In the example shown in Figure 3.2, we have considered a multi-cast frame f_1 transferred from v_1^e to v_2^e and v_3^e. Here, $f_1.\mathbf{ph} = \{f_1.ph_1, f_1.ph_2\}$, where $f_1.ph_1 = \{l_{1,4}, l_{4,2}\}$ and $f_1.ph_2 = \{l_{1,4}, l_{4,5}, l_{5,3}\}$. Furthermore, in the time-triggered communication, a frame must traverse each link according to some pre-calculated schedule. Therefore, $f_i.\mathbf{o}$ represents a set of values where a value $f_i.o^{l_{m,n}}$ corresponds to the frame offset in the link $l_{m,n}$. An application thus consists of an ordered chain of tasks and frame dataflow paths. In Figure 3.2, we can define two applications a_1 and a_2 for which the task chains can be written as $a_1.tc = \{\tau_1, f_1.ph_1, \tau_2\}$ and $a_2.tc = \{\tau_1, f_1.ph_2, \tau_3\}$ respectively.

The variables of an application that need to be obtained for a scheduling problem are the offsets of tasks and the transmission offsets of the frames on the links. We define this vector as $a_i.\mathbf{o}$. The latency of the application a_i is defined as the time delay between the beginning of the first task and the end of the last task in a task chain in an application instance. For each application, we denote the maximal allowed latency of an application as $a_i.lz$. Furthermore, we divide the applications into two categories, namely the *basic* applications and *plug-in* applications. The basic applications are the applications that provide basic functions of the ECU cluster (e.g., braking and steering control) and are deployed and fixed in the system. The functional correctness as well as the timing behaviour of these applications are rigorously tested and validated and the task and communication schedules cannot be modified. The plug-in applications are the plug-and-playable applications. These applications can be installed or updated after sales and their schedules can be changed provided that the timing constraints are satisfied. Therefore, we denote the possibility of re-scheduling an application as a boolean parameter $a_i.rs$. The parameters for an application $a_i \in \mathcal{A}$ include the period, task partition and mapping, the WCET of the tasks on the task side, the length and data flow paths of the frames on the communication side, the latency constraint and the possibility of re-scheduling. We represent this parameter set as $a_i.\omega$. The parameters of an application depends on the application software implementation, the hardware characteristics and can be obtained, provided the details of the software and hardware architecture are known. An application can then be represented by a tuple $\{a_i.\omega, a_i, \mathbf{o}\}$, and an application set $\mathcal{A}$ as $\{\mathcal{A}.\omega, \mathcal{A}.\mathbf{o}\}$.

The Scheduling Problem
The scheduling problem in this architectural setting then boils down to the problem of finding the task and frame offsets $\mathcal{A}.\mathbf{o}$ of all applications given the parameters $hw.\omega$ and $\mathcal{A}.\omega$. The hardware parameters $hw.\omega$ here include the network topology $G(\mathcal{V}, \mathcal{E})$ and the related device timing parameters. These timing parameters include: (i) network bandwidth bw, (ii) interframe gap ifg, i.e., the minimum idle time between two consecutive frame transmissions on a link, (iii) precision $sync$, i.e., the maximum clock-skew in the network, (iv) message packetization delay sd, i.e., the upper bound on the time taken by a sender ECU to packetize a data into a frame and subsequently make it available in the sending buffer, (v) message depacketization delay rd, i.e., the upper bound on the time taken by a receiver ECU to depacketize a frame to retrieve the data and make it available to the destination task, and (vi) switch propagation delay pd, i.e., the upper bound on the time taken by a switch to read the destination address, decide the link to which it must forward the frame and subsequently make it available in the corresponding sending buffer. In this context, the schedules for basic applications are synthesized offline and kept unchanged. Plug-in applications can be mapped on this schedule set either by (i) simple incremental scheduling, where all schedules of the existing applications are kept fixed and formulated into constraints, (ii) conflict-based re-scheduling, where only existing plug-in applications with conflicts with the new applications are rescheduled, or (iii) complete re-scheduling, where all schedules of the plug-in applications are rescheduled to accommodate the new ones. Both the simple incremental scheduling and the complete re-scheduling problem are well-studied in related works and can be formulated into a constraint programming problem, as discussed in details in the related works [59, 89, 99, 100]. The conflict-based scheduling will be explained in Section 3.4. Examples of constraints in this case include the following, where the detailed mathematical formulations can be found in Section 2.4 in Chapter 2:

- *Non-overlapping tasks and frames:* No two task instances mapped onto the same ECU should overlap with each other and no two frame instances on the same link should overlap with each other.
- *Path dependency in communication frames:* The forwarding of the frames in the network should follow the correct order of links in the corresponding virtual link.

- *Data dependency in applications:* In a task chain, the tasks and the data transmission should follow the correct temporal order specified by the data dependency.
- *Max. application latency:* The latency of an application should be smaller than a certain value.

3.3.2 Schedule Management Problem

As already mentioned, PnP feature of automotive embedded systems requires that the system possesses a certain degree of reconfigurability, where communication and computation resources need to be allocated for the new plug-in applications. In terms of a time-triggered architecture considered in this work, one important issue is to obtain the new schedules, where the reservation of the processor time and the communication bandwidth needs to be recalculated and deployed. In the context of this chapter, we assume that when a new application is installed/removed, or an update of the application changes its resource requirements, the parameters of the application $a_i.\omega$ are pre-calculated for the vehicle variant and deployed with the software. For example, the WCET of tasks can be obtained by either formal verification or by intensive testing and validation on actual hardware of the specific vehicle variant before the application is released. Based on this and the information on existing applications in the ECU cluster, a new schedule set needs to be obtained that can accommodate the new or updated application in the time-triggered system under consideration.

Towards this, there are several alternatives that can be considered. (i) One possible solution would be to obtain offline the schedules for a specific application and deliver it with the software. (ii) Another alternative would be to synthesize schedules offline and stored them on a server. Once an application is installed, the vehicle fetches the corresponding schedule set that matches the requirement. (iii) A third method is to synthesize the schedules online. This can be done either onboard the vehicle or on the server through cloud computing. (i) would not be applicable, since the schedule synthesis depends on the full knowledge on the hardware parameters and the parameters of all the existing applications in the target system. (ii) would be applicable in the case where an ECU cluster can accommodate all the related applications, i.e., the basic and plug-in applications. In this case, a 'Master' schedule set can be synthesized offline with all applications and the schedules for a particular scenario can be generated by removing the applications not needed. However, as mentioned in Section 3.1, one of the motivation for plug-and-play is that new applications are constantly developed, but the E/E architecture of the vehicle remains relatively static for the life cycle of the vehicle. The number of available applications can easily exceed what a system can accommodate. In that case, multiple schedule sets need to be synthesized and maintained to cover all possible combinations of applications that can be accommodated by the system. The number of schedule sets can easily explode as the number of applications increases. In this case, method (ii) would require the server to maintain a huge number of schedule sets, which adds to the complexity and also the time needed to retrieve a valid schedule set. Furthermore, this might not be necessary if some of the application combinations do not appear in practice. Method (iii), however, results in the online solving of the scheduling problem, which might lead to much longer time for obtaining the schedules.

The problem this chapter addresses is how to efficiently obtain new schedules online and manage the schedules, while considering the requirements like low latency for obtaining the schedule configuration in a plug-and-play scenario. This can be done by combining (ii) and (iii), exploiting the advantages on both sides and by utilizing the storage and computation capacity on the server.

3.4 Approach

In this section, we describe the proposed software framework for the schedule management. We first start by the definition of the request and configuration of the schedules. Then the client-server model of the software and each individual software component are explained. Finally, we explain the management flow of the framework.

3.4.1 Request and Configuration

We firstly define a configuration format to serve as the interface between different software components and between the client and the server. The configuration format termed TTCON is an XSD (XML Schema Definition) and can be used to represent the system described in Section 3.4. This format contains mainly three parts, namely the hardware specific parameters $hw.\omega$, the application parameters $\mathcal{A}.\omega$ and the application schedules $\mathcal{A}.\mathbf{o}$. The TTCON format can be used to represent a valid schedule *configuration* or a *request*. In a configuration, the task and communication schedules $\mathcal{A}.\mathbf{o}$ have valid values for all applications. In a request, the schedules of the applications are left blank, indicating that they need to be determined. We define the set of applications in the current configuration as $\mathcal{A}_o = \mathcal{A}^b \cup \mathcal{A}_o^a$, which is a combination of the set of basic applications $\mathcal{A}^b$ and plug-in applications $\mathcal{A}_o^a$. The new set of applications that are requested to be activated is denoted as $\mathcal{A}_n = \mathcal{A}^b \cup \mathcal{A}_n^a$. Note that we assume in this work, the basic applications cannot be installed or deleted in a plug-and-play fashion and the schedules of them are kept unchanged. In the case of request for a new schedule set, a request C_r consists of the hardware specific parameters $hw.\omega$, the application parameters of $\mathcal{A}_n = \mathcal{A}^b \cup \mathcal{A}_n^a$ and the application schedules of $\mathcal{A}_n$, where all the values of the schedules are blank. The configuration and the request can then be passed between the software modules, where the schedules of the whole application set are determined. This will result in a new configuration C_n, which can be used for deployment.

3.4.2 Client-Server Architecture

The proposed schedule management framework follows a client-server based architecture. The client can be mapped on an ECU onboard the vehicle and the server can be mapped on a server computer that can be maintained by the car manufacturer. The client and the server are connected through the Internet. Both the client and the server consist of four components, namely the *management module*, the *configuration pool*, the *web module* and the *synthesis module*.

The management module is responsible for the whole process control. The configuration pool stores valid configurations, which can be reused. The web module is responsible for the communication between the server and client. The client can send requests to the server, while the server can send result configurations back to the client. If a new schedule configuration is obtained locally, the client can also send it to the server to update the server configuration pool. The web module uses websocket secure, which has a layer of Secure Sockets Layer/Transport Layer Security (SSL/TLS) and therefore provides a certain level of security to the client-server communication. The synthesis module provides the ability to synthesize or partially synthesize schedules online. Here, we use a four-step strategy, starting from a simple incremental design to a complete rescheduling of all plug-in applications. The software architecture of the proposed framework is shown in Figure 3.1. The aforementioned components will be explained in detail in the rest of this section.

3.4.3 Configuration Pool

The configuration pool is used to store and manage valid configurations. There are mainly two access methods of the configuration pool, namely (i) retrieving a configuration and (ii) updating the pool. On both the client and the server, a request can be passed to the configuration pool to check if a valid configuration already exists. This is done by comparing the hardware parameters $hw.\omega$ and the application parameters $\mathcal{A}.\omega$ in the request C_r and the possible candidate configurations in the pool. If the set of applications in a request matches exactly or is a subset of the application set of a configuration in the pool, the specific configuration can be retrieved and utilized. The later case can be dealt with by removing the parameters and schedules of the applications that are not in the request. Configurations obtained through synthesis can also be added to the configuration pool so that it can be reused later. Considering the limited storage size on the client side, only a limited number of configurations are stored on the client. The configuration pool can be updated and managed, e.g., by keeping the latest or the most frequently used configurations. To efficiently utilize the capacity of the pool, if a new valid configuration is added to the pool, it is firstly checked if the applications contained is a superset of any of the configurations in the pool. If such a case is true, the new configuration simply replaces the existing one and inherits its properties (e.g., the frequency of reuse). On the server side, it is reasonable to assume that a more powerful computer with much larger storage size is used, where many more configurations can be stored. In this case, schedules can be reused between different vehicles of the same variant. The configuration pool thus offers the possibility of reutilizing previous configurations so that the online synthesis effort can be reduced. In addition, schedules can also be synthesized offline and stored in the configuration pool of the server.

3.4.4 Web Module

The web module is responsible for the communication between the server and the client and uses the websocket secure protocol. Websocket is a protocol commonly used in the implementation of modern web browsers and web servers and is already beginning to be applied in the connected car area for transmission of information. Compared to the Hypertext Transfer Protocol (HTTP) protocol, websocket also allows full-duplex communication and therefore is suitable in this case for the server to send the result configuration back to the client. The websocket secure provides additional security for the communication between the server and the client with an additional layer of SSL/TLS, where certificates and encrypted data transmission are used. In this module, we define the following communication between the client and server: (i) *Request*: The client opens a connection to the server and sends the request file C_r and the current configuration C_o to the server. (ii) *Response*: The server sends back a response to the client. This response can either contain a valid configuration file fetched from the configuration pool or synthesized by the server, or a request denial, where the server is unable to obtain a valid configuration (e.g., the problem is infeasible). (iii) *Abort*: The client can also notify the server to stop trying to obtain a configuration, when a valid schedule is already obtained locally on the client side. (iv) *Update*: If the schedules are synthesized locally, the Abort method is followed by an update method, where the client sends the local result to the server to update the configuration pool. In the Request, Response and the Update method, a configuration needs to be sent. If the configuration file is larger than the maximum TCP packet size, it will be fragmented at the sending side and assembled at the receiving side. A connection is always started by the client and closed by the client once it has received a Response from the server or after sending the Abort and Update message to the server.

3.4.5 Synthesis Module

The function of the synthesis module is to find a feasible schedule set for the tasks and messages. It utilizes a constraint programming solver, and is able to process the request, formulate the constraint programming problem, solve the problem and generate the corresponding schedule configuration. In this module, we introduce a four-stage synthesis strategy. In the first stage, we use a simple incremental design, where the schedules of all remaining existing applications are kept unchanged and the new applications are mapped

Algorithm 1 Four-stage scheduling scheme

```
Input: $C_o$, $C_r$
Output: $C_n$
Initialization: $stage = 1$, $C_n = \emptyset$
 1: while $stage \leq 4$ do
 2:     $[A_k, A_u] =$ preProcessing$(C_o, C_r, stage)$
 3:     $M =$ modelFormulation$(C_o, C_r, A_k, A_u)$
 4:     $[S, isSolved] =$ solver$(M)$
 5:     if $isSolved == true$ then
 6:         break
 7:     end if
 8:     $stage = stage + 1$
 9: end while
10: if $isSolved == true$ then
11:     $C_n =$ generateConfiguration$(S, C_o, C_r)$
12: end if
13: return $C_n$
```

upon it. If a feasible solution cannot be found, we use a method with task conflict based backtracking as a second stage, where the existing plug-in applications which share common tasks with new applications are also rescheduled. In the third stage, a scheme with resource conflict based backtracking is employed, where the existing plug-in applications which have tasks mapped on common ECUs with the tasks of the new applications are rescheduled. If all previous stages fail to find a feasible solution, a complete rescheduling of all the applications except the basic ones is carried out. One objective for the four-stage synthesis strategy is to minimize the disturbance to the schedules of existing applications. Compared to this approach, using only simple incremental design would seriously reduce the chance of accommodating new applications, even if the system has enough resources. On the other hand, rescheduling all existing plug-in applications every time would possibly result in much longer synthesis time and disturb the existing applications, which may not be necessary. In the introduced strategy, we are essentially making a trade-off between the chances of accommodating new applications on one side and disturbance to existing applications and the synthesis time on the other. As we move from stage one to stage four, more existing applications are rescheduled, and therefore both the chances of accommodating the new applications and the synthesis time increase. Here, we use the conflict of tasks and computation resources as a metric for selecting the existing applications to reschedule. The reason is that the existing applications with common tasks and common task mappings are more likely to constrain the possibility of accommodating new applications. Here we do not consider frame mapping conflict since compared to the ECUs, the network might be relatively less loaded [100].

Algorithm 1 describes this four-stage scheme. This approach adopts an iterative scheme (Line 1-9). Here *stage* represents the current scheduling scheme used, where 1 to 4 denote respectively the simple incremental design, the task conflict based rescheduling, the resource conflict based rescheduling and the complete rescheduling. This difference is made

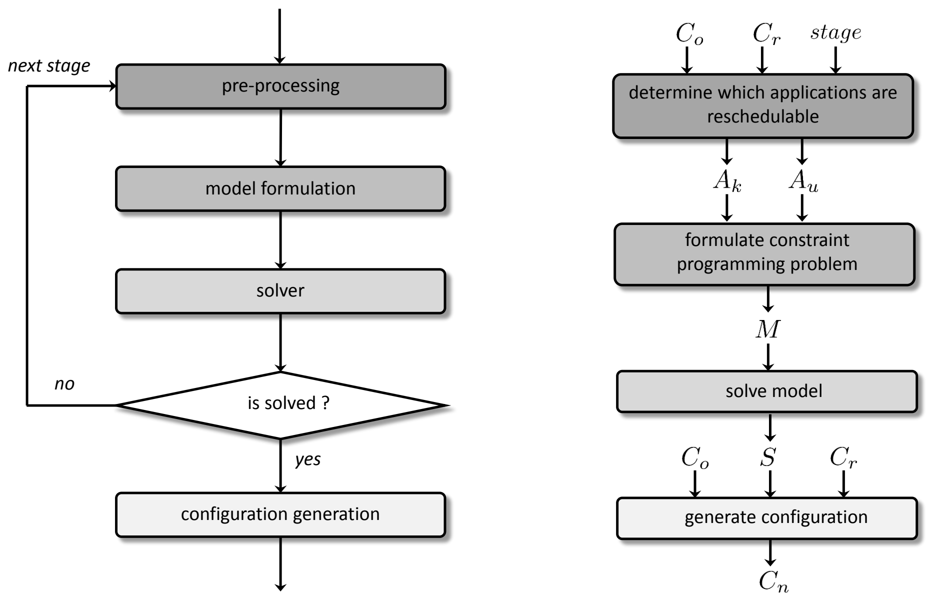

Figure 3.3: The flow diagram for the synthesis module. The left hand side shows the flow diagram for the synthesis module. The right hand side shows the processes and the input and output.

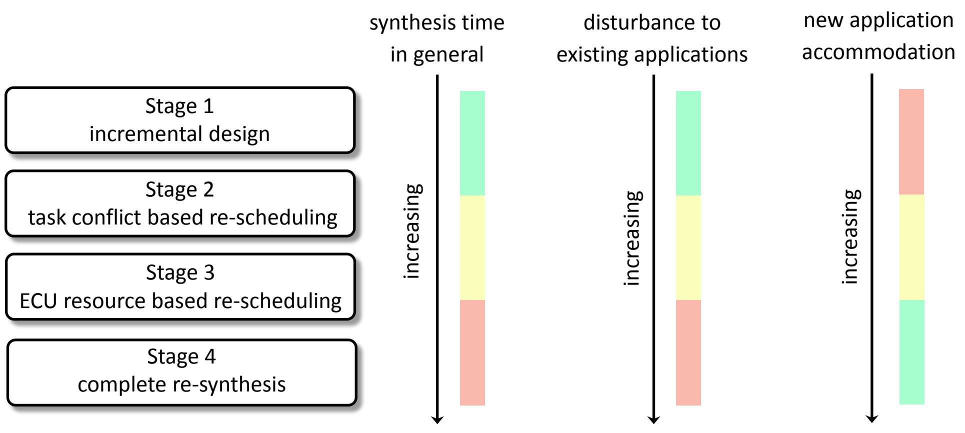
synthesis time in general
disturbance to existing applications
new application accommodation
Stage 1 incremental design
Stage 2 task conflict based re-scheduling
Stage 3 ECU resource based re-scheduling
Stage 4 complete re-synthesis
increasing
increasing
increasing

Figure 3.4: This figure shows the comparison between the four stages. From stage 1 to stage 4, the synthesis time, the disturbance and the chance of accommodating new applications all grow. For the first two, this increase has negative effects and for the third one, the increase has positive effects.

in the pre-processing stage (Line 2), where the applications in $\mathcal{A}_n^a$ is divided into two sets: (i) the non-reschedulable set $\mathcal{A}_k$ and the reschedulable set $\mathcal{A}_u$. $\mathcal{A}_k$ and $\mathcal{A}_u$ can be calculated for stage 1 to stage 4 according to Eq. (3.1) to Eq. (3.4) respectively. Here, $\backslash$ represents the set difference and for example $\mathcal{A}_n \backslash \mathcal{A}_k$ represents the applications that are in $\mathcal{A}_n$ but not in $\mathcal{A}_k$. In the case of stage 2 and stage 3, the application set $\mathcal{A}_\tau$ with task conflict and $\mathcal{A}_E$ with resource conflict can be obtained using Eq. (3.5) and Eq. (3.6). Therefore, the corresponding $\mathcal{A}_k$ and $\mathcal{A}_u$ are generated and used by the model formulation component (Line 3) and the solver component (Line 4) to synthesize the schedules. If a valid solution can be obtained (Line 5) in one stage, the configuration is generated (Line 11) and returned. Otherwise the algorithm moves to the next stage (Line 8). If no feasible schedule set can be found in stage four, the synthesis module will not return a valid configuration.

$$\mathcal{A}_k = \mathcal{A}_o \cap \mathcal{A}_n, \mathcal{A}_u = \mathcal{A}_n \backslash \mathcal{A}_k \tag{3.1}$$

$$\mathcal{A}_k = (\mathcal{A}_o \cap \mathcal{A}_n) \backslash \mathcal{A}_\tau, \mathcal{A}_u = \mathcal{A}_n \backslash \mathcal{A}_k \tag{3.2}$$

$$\mathcal{A}_k = (\mathcal{A}_o \cap \mathcal{A}_n) \backslash \mathcal{A}_E, \mathcal{A}_u = \mathcal{A}_n \backslash \mathcal{A}_k \tag{3.3}$$

$$\mathcal{A}_k = \mathcal{A}^b, \mathcal{A}_n = \mathcal{A}_n^a \tag{3.4}$$

$$\mathcal{A}_\tau = \{a_i | a_i \in \mathcal{A}_o^a \cap \mathcal{A}_n^a \wedge \underset{a_j \in A_n^a \backslash A_o^a}{\exists} \ \underset{\substack{\tau_k \in a_i.tc \\ \tau_l \in a_j.tc}}{\exists} \tau_k = \tau_l\} \tag{3.5}$$

$$\mathcal{A}_E = \{a_i | a_i \in \mathcal{A}_o^a \cap \mathcal{A}_n^a \wedge \underset{a_j \in A_n^a \backslash A_o^a}{\exists} \ \underset{\substack{\tau_k \in a_i.tc \\ \tau_l \in a_j.tc}}{\exists} \tau_k.E = \tau_l.E\} \tag{3.6}$$

3.4.6 Request-based Schedule Management Flow

Figure 3.5 and Figure 3.6 shows respectively the program flow of the client and the server. Each session is triggered by the request from the client for obtaining a new schedule configuration. The management module firstly generates a request C_r based on the set of applications to be activated along with the hardware specific parameters and the parameters of the basic applications. Then it checks the local configuration pool for an existing configuration. If no valid configuration can be found, it simultaneously starts the web module and the synthesis module and waits for the result from both. The management module, the web module and synthesis module are mapped on different threads so that they can be executed asynchronously. The web module then initiates a connection to the server, sends the request and waits for a response. The synthesis module then tries to synthesize a feasible schedule set according to the four-stage strategy explained in Section 3.4.5. If a valid schedule set can be synthesized, a new configuration is generated and returned to the management module. Otherwise, the synthesis module will deny the request. If a valid configuration is returned from either side, the management module tries to stop the other

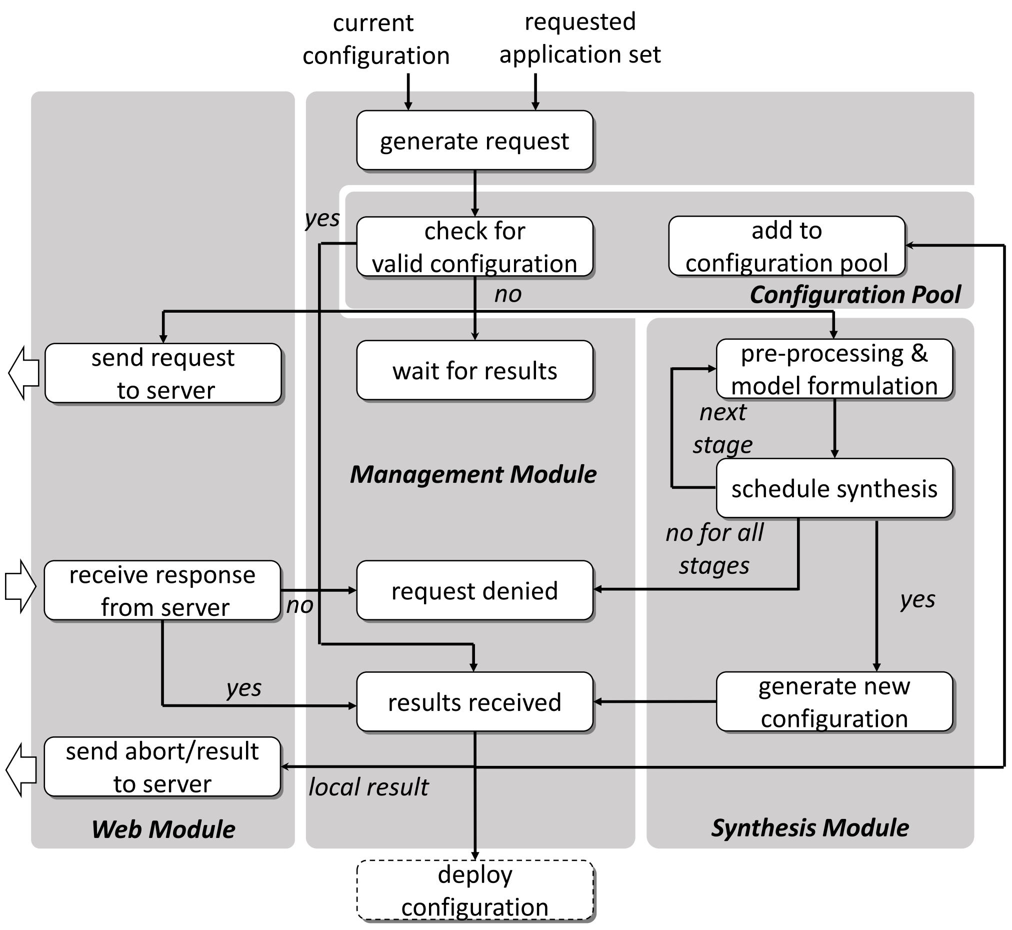

Figure 3.5: Management flow on the client side.

side, saves the new configuration into the configuration pool and passes the new configuration for reconfiguration. In the case where a result is obtained locally, the management module will use the the Abort and Update method in the web module to stop the effort of the server and send it the local result so that the configuration pool on the server can be updated. When both sides deny the request, the management module will deny the request and does not allow for a reconfiguration.

On the server side, the session starts when a request from the client is received by the web module. It then checks its configuration pool and if a valid configuration can be found, it sends the result back to the client. Otherwise, it will start the synthesis module and try to find a feasible schedule set. Once a valid configuration is obtained, the server sends the result back to the client through the web module Response method. If no feasible schedule set can be obtained, it will send a request denial to the client also using the Response method. If an Abort message followed by an Update message is received from the client, the server tries to stop the current process and adds the configuration result into its configuration pool.

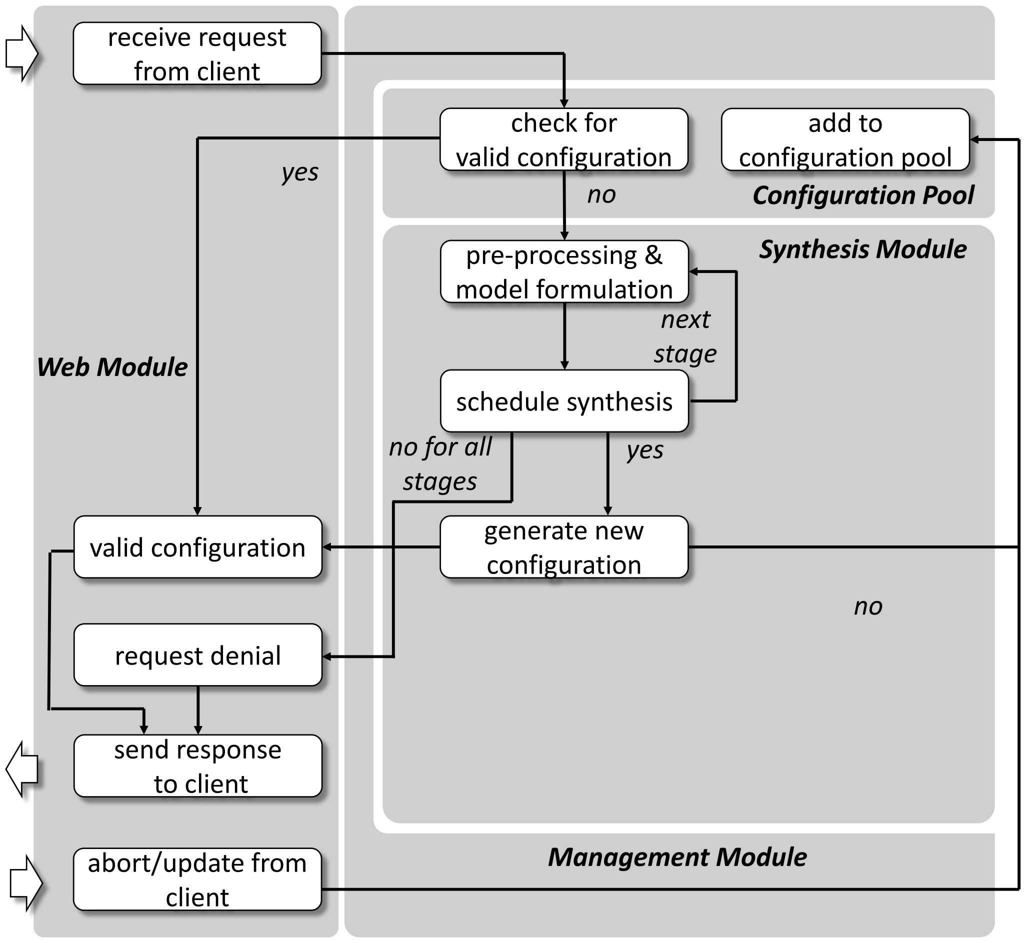

Figure 3.6: Management flow on the server side.

3.5 Experimental Results

Towards the evaluation of the applicability and performance of the proposed software framework, we have implemented the client on a Raspberry Pi 2 Model B and the server on a laptop computer. The client side has a 900 MHz quad-core ARM Cortex A7 processor and Raspbian OS. The server is running on a Windows machine with an Intel Core i7-4600 CPU. Both sides are connected to the Internet through WLAN connection. One of the implementation challenges here is the portability of the constraint solver on the Raspberry Pi platform. Both on the client and the server, we use Gecode [105] as the constraint solver. We have evaluated several choices of solver including Gecode, Gurobi and Z3. Gecode is the only solver that can be compiled on the Raspberry Pi and is in general faster than Gurobi and Z3 for this particular constraint programming problem. The websocket secure implementation from [106] is also used as part of the implementation of the web module.

In the experiments conducted, we use a synthetic case study. The hardware architecture consists of 10 ECUs connected by 4 switches. 100 applications are randomly generated.

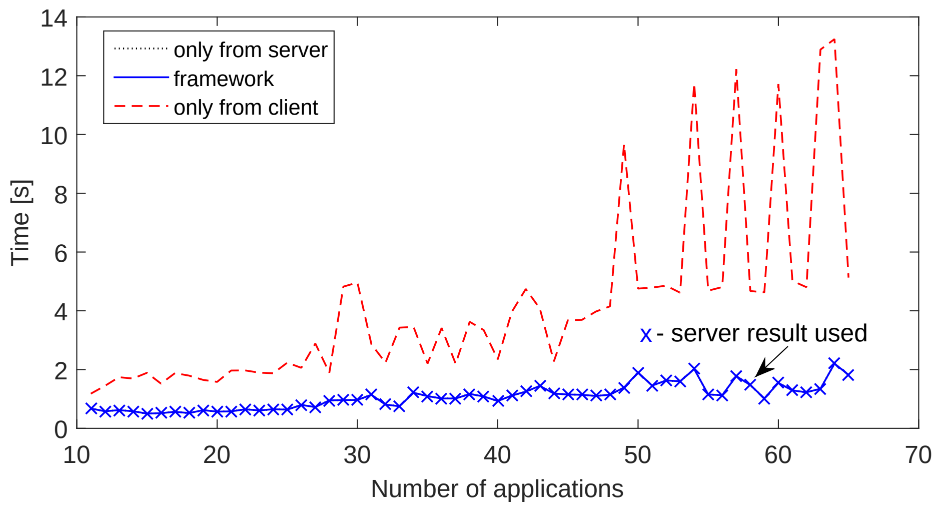

Figure 3.7: Schedule synthesis time of the proposed framework for a sample request series, compared to only using the cloud-based result or the result on the client in the case of 0 seconds additional overhead provision.

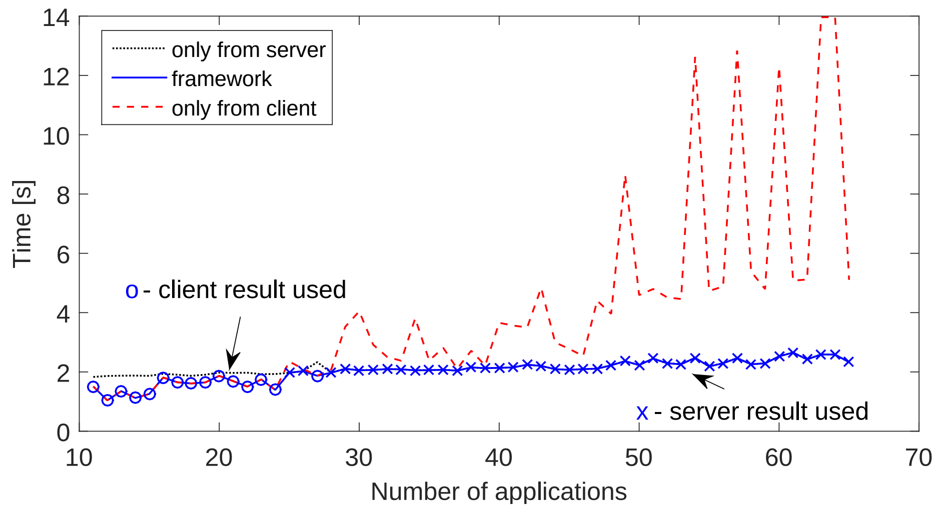

Figure 3.8: Schedule synthesis time of the proposed framework for a sample request series, compared to only using the cloud-based result or the result on the client in the case of 1.5 seconds additional overhead provision.

Each can have a length of two to seven tasks (a message is sent between any two consecutive tasks). The periods of the applications are randomly chosen from a pool of 5, 10, 15 and 20 ms. The WCET of the tasks are between 300 us and 700 us. The frame lengths are selected from discrete values of 64, 80, 96, 112 and 128 bytes. Different applications can share common tasks and messages. We consider 10 applications as basic applications and their task and message schedules are synthesized offline and kept unchanged during the whole experiment. The other 90 applications are plug-in applications, which can be installed on

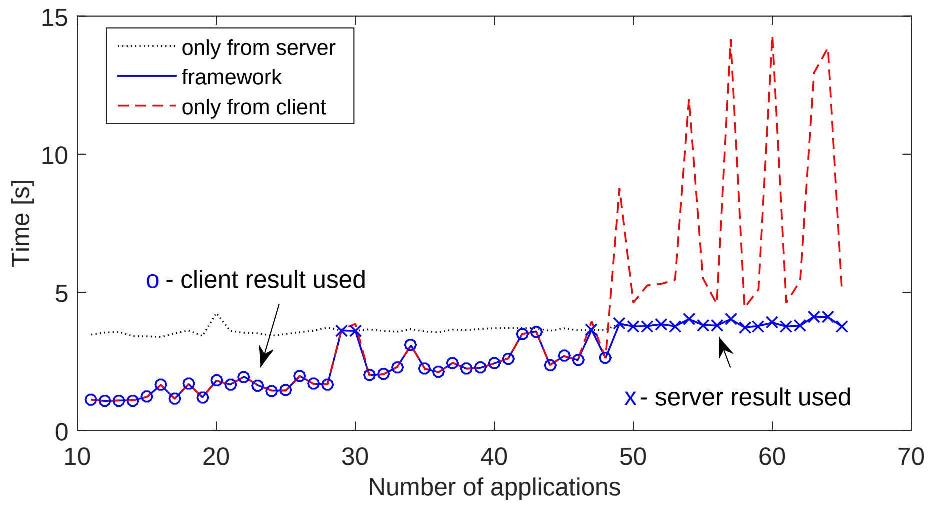

Figure 3.9: Schedule synthesis time of the proposed framework for a sample request series, compared to only using the cloud-based result or the result on the client in the case of 3 seconds additional overhead.

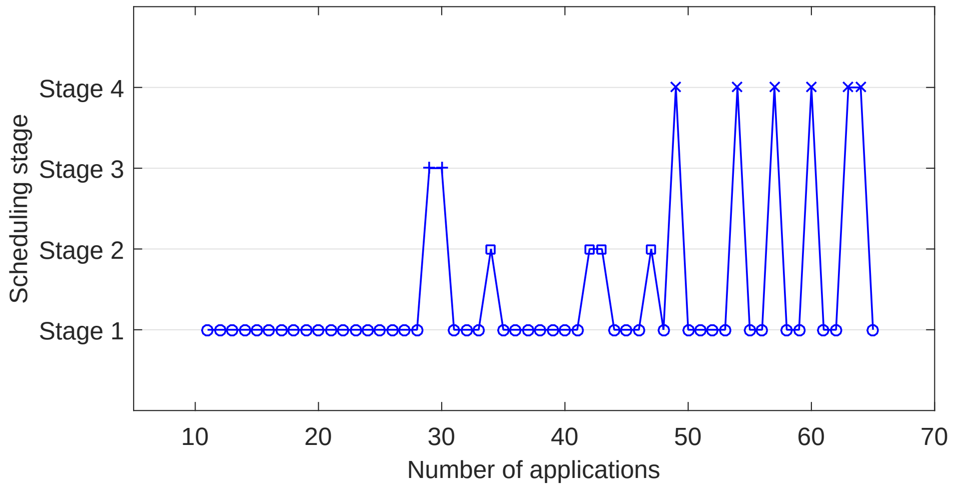

Figure 3.10: Scheduling stages used for a sample request series.

the system. In this experiment we consider the case where a system starts with the basic applications and the plug-in applications are mapped incrementally. For this purpose, we generated 20 different request series of adding the plug-in applications. For each request series, we stop adding applications when a feasible schedule set cannot be found or by a timeout of the solver (60 seconds for each scheduling stage on the laptop computer).

In our implementation, we have only used the security mechanism of the websocket secure for the communication. In the final application scenario, however, more complex authentication and security protocols could be required, if the schedules are obtained from the cloud, which will introduce additional timing overhead. Therefore, we have considered three different cases, where an additional delay of 0, 1.5 and 3 seconds is introduced on the server respectively. We use this additional delay as a representation of the overhead provision for these additional mechanisms that could be required.

Figure 3.7, Figure 3.8 and Figure 3.9 show the result from one request series for the case of additional server delay of 0, 1.5 and 3 seconds, respectively. The blue solid line shows the time required by the framework in each step in the series against the number of applications in the system. The round symbol and the cross symbol denote that the result is taken from the client and the server respectively. In comparison, the red dashed line shows the time taken for synthesizing the schedules only on the Raspberry Pi and the black dotted line shows the time taken for synthesizing the schedules only on the server through cloud-computing. The data for these two are generated by not stopping the local synthesis module and turning off the Abort and Result method in the web module, so that both the time taken by local synthesis and cloud-based synthesis is recorded. We can observe from the three figures that the time taken on the Raspberry Pi (red dashed) is quite small (1 to 2 seconds) when there are few applications mapped on the system and increases considerably (10 to 14 seconds) when the number of applications grows. On the other hand, the time taken by only using the cloud-computing fluctuates around a certain level depending on the additional delay. The reason for this is that the Raspberry Pi has a limited computational capacity and therefore the time to obtain a schedule is very sensitive to the complexity of the synthesis problem. For example, in the large fluctuations between 48 and 65 applications in the red dashed line, the peaks and the troughs denote different scheduling stages used, which can be observed from Figure 3.10. On the other hand, the laptop is much faster and therefore the synthesis time is relatively small and the time taken by cloud-based synthesis is dominated by the web framework time (e.g., establishing connections, transmission of request and response and the overhead provision). In Figure 3.7, when the cloud-based synthesis has zero additional delay, the framework always uses the result from the server, which is considerably faster. In Figure 3.8 and Figure 3.9, where additional overhead provision is considered, in some of the cases, the client result is faster. Especially in the case of Figure 3.9, when the number of applications is less than 40, the result is obtained locally on the client for the majority of the cases. However, when the system size gets larger and the problem becomes more complex, the server result is used, avoiding the long time of the local synthesis, as shown by the spikes in the red dashed line between 48 and 65 applications. Therefore, by using the proposed framework, we can take the advantage of both the local synthesis and cloud-based synthesis. In addition, sometimes there could be uncertainties in the network latency in the cloud environment (e.g., the spike at the size of 20 in the black dotted line in Figure 3.9 or at the size of 27 in Figure 3.8). In such a case, the local synthesis result will be used if it is faster.

Figure 3.10 shows in addition the scheduling stages used in the four-step strategy in the same request series. The round, square, plus and cross symbols denote the stage 1, 2, 3 and 4 of the schedule synthesis scheme respectively. It can be observed from the figure that when the number of applications are small (until 28), new plug-in applications can be accommodated by stage 1, i.e., simple incremental design. However, the next application

	Stage 1	Stage 2	Stage 3	Stage 4	Total
Delay = 0 s					
Client	0.19%	0.00%	0.00%	0.00%	0.19%
Server	82.72%	5.67%	6.51%	4.91%	99.81%
Delay = 1.5 s					
Client	29.84%	0.19%	0.00%	0.00%	30.03%
Server	53.07%	5.48%	6.51%	4.91%	69.97%
Delay = 3 s					
Client	67.33%	2.36%	0.00%	0.00%	69.69%
Server	15.58%	3.31%	6.51%	4.91%	30.31%
Total	82.91%	5.67%	6.51%	4.91%	100.00%

Table 3.1: Percentage share of the scheduling stages and whether the client or server result is taken for all the steps in the 20 request series.

cannot be added to the system using stage 1, and therefore the synthesis module goes to stage 3 by rescheduling existing applications with the common resources. Afterwards, as the number of applications increases, it is becoming more difficult to use simply stage 1, and therefore stage 2, 3 and 4 are employed to synthesize the schedules. Combining with the red dashed line in Figure 3.7 to Figure 3.9, it can also be observed that the complexity and the time required by the synthesis also increases from stage 1 to stage 4. Furthermore, from the size of 30 to 65 applications, we can see that sometimes a stage 1 scheduling becomes possible again after a higher stage rescheduling. This shows that the four-stage scheduling scheme can offer a dynamic balance between the chance of accommodating new applications and the rescheduling effort.

Table 3.1 summaries the scheduling stage used for all the steps in the 20 request series for different additional server overhead. It also shows whether the client result or the server result is used. Firstly, we can observe from the table that in the majority of the steps (82.91%), a new application can be accommodated using scheduling stage 1. However, stages 2, 3, 4 are also employed in some steps. Secondly, in the case where additional server delay is zero, almost all results are obtained through the cloud. As the additional delay increases, increasingly more results can be obtained from the client. In the case where the additional delay is 3 seconds, in 69.69% of the steps, the result can be obtained from the client, mostly the steps using stage 1.

Figure 3.11 shows the scheduling stages used, which result is taken and the time taken for obtaining the schedules in most of the steps in the 20 request series for the case of 1.5 seconds additional server delay. Several steps, where the time is much longer than 20 seconds, are not included in the figure for the purpose of clear illustration. The result for synthesizing on the client, using the proposed framework and using only the server is shown in (a), (b) and (c) respectively. From plot (a), we can see that in general, the schedule synthesis time increases from stage 1 to stage 4. Please note that the time for a specific stage also includes the time taken by all previous stages where no feasible schedules can be

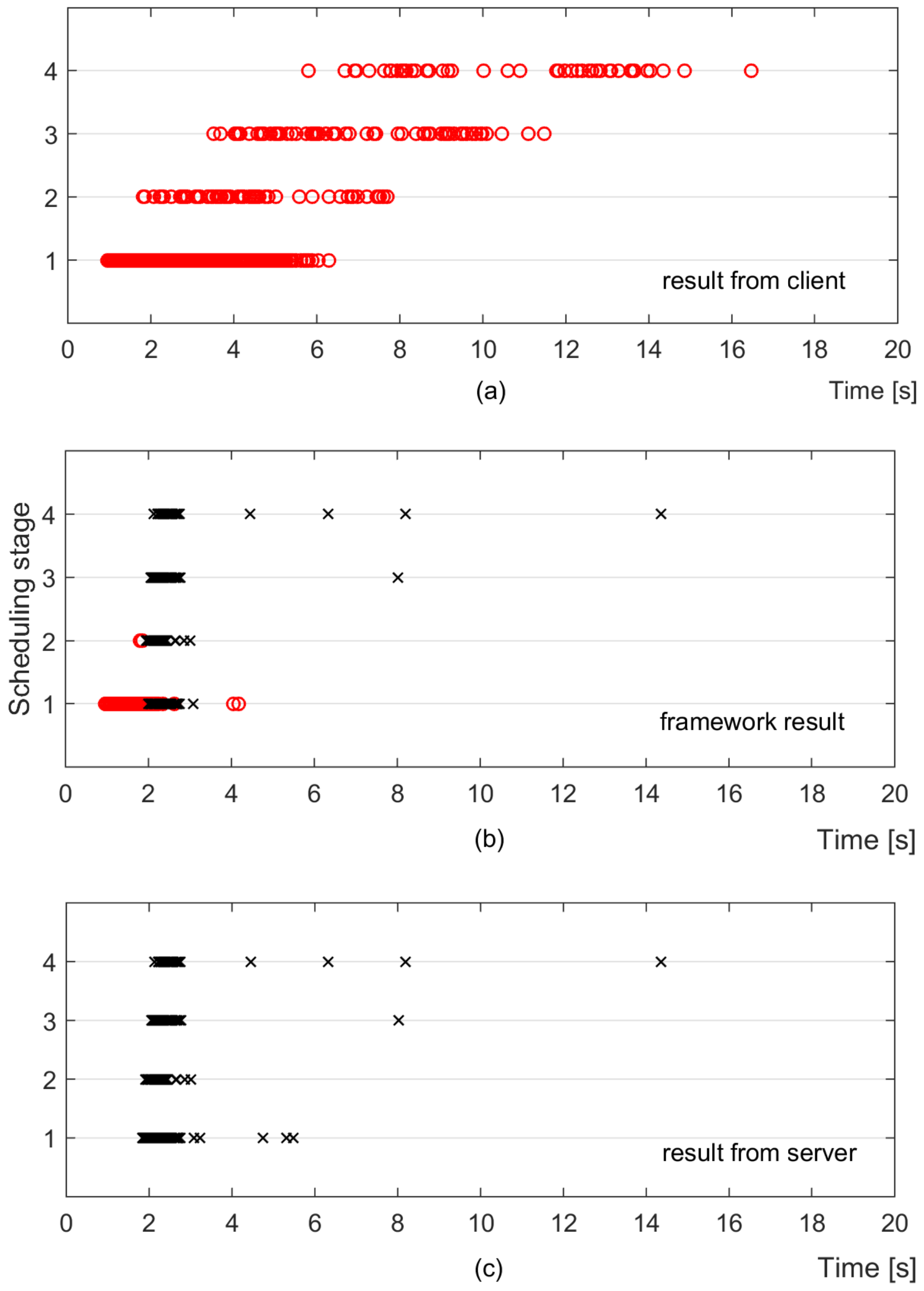

Figure 3.11: Comparison of the synthesis time between the proposed framework and only using the server result or only using the client result in the case of 1.5 seconds additional overhead.

found. Figure 3.11 (c) shows that by only using the cloud-based synthesis, the time taken by stage 3, 4 and part of stage 2 is considerable reduced. However, the time taken by part of stage 1 is increased, which can be accounted for by the web framework time. The result in plot (b) shows that the proposed framework applies the reduced time from the cloud for stages 2, 3 and 4, while preserving the fast results of the local synthesis in stage 1.

In this experiment we have used a Raspberry Pi as a representative ECU. It should be noted that actual ECU hardware might differ from the hardware used in the experiments. For example, there are now faster ECUs available and in the future the ECUs are going to become more powerful. Our intention here is to illustrate the framework and provide a proof of concept. The actual numbers in the final application scenario depends heavily on the hardware platform. However, the experimental results have shown that the proposed framework can always benefit from both sides and handle different situations robustly.

3.6 Concluding Remarks

In this chapter, we propose a schedule management framework for a schedule reconfiguration in the PnP scenario in future automotive embedded systems. The framework enables online schedule generation and management for an Ethernet-based time-triggered system. It utilizes the Internet access to allow a mixture of schedule generation and storage both locally and in a cloud-computing fashion.

One possible future work in this direction is to exploit the multi-core architecture of the processors to place different scheduling schemes parallel so that the schedule synthesis time can be further reduced. Furthermore, approaches in extensibility-aware scheduling can be utilized here so that more applications can be accommodated using incremental scheduling. This chapter addresses the online schedule generation and management problem. However, there are still many challenges that need to be addressed for the PnP capability and reconfigurability of future automotive embedded systems. Besides those mentioned in Section 3.1, one main issue that still needs to be addressed in the online reconfiguration of schedules is the safety issue. Although the time-triggered schedules are by design correct, we understand that for safety-critical applications, currently intensive testing and validation is done offline for both the functional and timing behaviour by the automotive industry. However, formal verification and methods might provide some solution to this problem in the future.

Chapter 4

Designing Resource-aware CPS over Hybrid Communication Networks

This chapter introduces a scheme towards resource-efficient CPS over hybrid communication protocols. Resource-aware design has been drawing increasingly more attention in the context of CPS during the recent years. The communication resource is one of the most important resources in a distributed embedded system and easily becomes a bottleneck problem as the size of the system increases. In this chapter, we address the problem of designing resource-aware CPS over the hybrid communication bus. Such a bus protocol offers both TT and ET communication. The TT communication offers higher timing predictability, which can potentially be translated into better control performance. However, such resource is often quite limited in safety-critical systems. It has been shown that it is possible to reduce the usage of TT communication by letting a control application use it only when necessary and use ET communication when it suffices [56]. Towards this, we introduce in this chapter, a resource-aware switching scheme for distributed embedded control applications and illustrate our method using the FlexRay communication protocol commonly used in the automotive domain. In the proposed approach, we use a combination of TT and ET communication to reject a disturbance and the strategy allows a control application to reside on the TT communication for an amount of samples that would be optimal for the overall control performance. This involves both an offline control design method that optimizes the control performance and guarantee the switching stability and an online scheduling algorithm to dynamically determine the allocation of the TT communication to multiple control applications at run-time. The experimental results show that the proposed approach can achieve good control performance with reduced consumption of TT communication bandwidth.

Chapter outline: This chapter is divided into seven sections. A general introduction is provided in Section 4.1, which explains the motivation, overview and contributions of this chapter. This is followed by the related works in Section 4.2. Section 4.3 provides the background and problem formulation. Then Section 4.4 explains the proposed control design method. This is followed by the presentation of the resource-aware switching scheme in Section 4.5, including the switching scheme, the offline generation of switching sequence model, the online scheduling algorithm and a middleware-based solution to address the implementation challenges. Section 4.6 presents the experimental results and Section 4.7 provides the concluding remarks.

4.1 Introduction

Modern industrial embedded systems are experiencing rapid increase in size and complexity. In the automotive domain, for example, there are already up to 100 ECUs in a premium vehicle and software for such a vehicle contains 100 millions lines of code [2]. As increasingly more functions are mapped on the E/E architecture, providing adequate embedded resources like communication resource, computation resource and memory resource is becoming a bottleneck problem. Therefore in the recent years, engineers and researchers have become conscious, that systematic approach for resource-efficient design of embedded systems is necessary [73, 107]. Especially the efficient utilization of the communication resource is an important issue, since new processing units can be added to the system or existing ones replaced by more powerful ones, while adding or upgrading one communication bus introduces a lot of development costs for the design, testing and validation process.

The wide variety of functional/timing requirements in the modern distributed systems (e.g., automotive E/E architecture) had made hybrid protocols like the FlexRay bus an attractive option as the communication medium. Such hybrid protocols offer both TT and ET communication, thus making it suitable for systems with mixed-criticality applications. In this work, we consider a system consisting of multiple distributed embedded control applications over the FlexRay bus [8, 9]. The FlexRay protocol offers the TT and ET communication respectively on the static segment and dynamic segment. Messages assigned to the TT communication are transmitted in pre-defined time slots and thus the timing is temporally predictable. For the ET communication, the timing of the message transmission varies depending on the network load at run-time and is thus much less predictable. It is therefore difficult to synchronize the communication with the software tasks on processing units, hence often results in much larger end-to-end delays.

Embedded control applications are implemented as software components that run on the processing units. In a distributed system, multiple dependent tasks are mapped on different processing units connected by a communication bus. Therefore, the task and message scheduling determines the timing properties like the sampling period and the sensor-to-actuator delay, which in turn influence the performance of the control applications. In general, if the sensor-to-actuator delay is small, it is possible to design controllers to achieve better control performance (e.g., smaller settling time). On the other hand, larger sensor-to-actuator delay might limit the design choices for the controllers and thus constrain the achievable control performance. Therefore, deterministic communication is desirable for distributed control applications to achieve a better control performance.

However, the deterministic TT communication resource that can be provided is usually limited. In FlexRay for example, the TT slots are considered more 'expensive' than the ET slots due to multiple reasons. Firstly, the design process in the automotive industry usually follows an iterative scheme, and during each design iteration, new functions can be mapped onto the system, while the current bus configuration and the message-to-slot mapping are kept unchanged for as much as possible. Hence, in each design iteration, not only the current requirements, but also those for possible future applications need to be considered. This design paradigm, considering the extensibility of the bus configuration, requires the saving of TT slots as much as possible. Secondly, the TT communication has a poor bandwidth utilization. The length of TT slots are often configured with respect to the whole system and therefore tend to be large enough so that different types of messages can fit in. The choice of multiplexing data in one slot is not always available. If such a slot is used to transmit a few bytes, the rest of the bandwidth is wasted. Furthermore, if a time-triggered slot is assigned, even if there is no data to be send in the corresponding cycle, the slot is still occupied. Therefore, from a resource saving perspective, it is desirable to use a TT slot only if necessary.

Therefore, there have been some initial works [56, 108] on resource-aware design through switching the controllers between the TT and ET communication, thus achieving savings of expensive TT resource while providing better control performance than using purely ET communication. However, there are still many challenges to be addressed and places to be optimized towards designing an efficient, safe and practical switching scheme. Firstly, the switching scheme can be optimized to achieve better control performance. Secondly, a systematic design method for such a control scheme guaranteeing the safety while optimizing the control performance and resource utilization is necessary. Furthermore, implementation hurdles introduced by the protocols and the commercial-off-the-shelf (COTS) tools need to be addressed for such a scheme to be practical.

Here we address the aforementioned challenges and propose a resource-efficient switching scheme that allows a control application to dynamically switch between the TT and ET communication. The TT communication resource can thus be shared between multiple control applications. To enhance the overall performance, we allow a control application to switch to TT communication and resides there for several samples and switch back to ET communication before it is settled so that the TT resource can be released earlier to be utilized by other control applications. We propose an online scheduling algorithm that decides at run-time the resource allocation and switching sequence so that the overall control performance is optimized. Furthermore, we introduce a automated design algorithm to design both controllers in the TT and ET case so that the switching stability is guaranteed and the combined performance is optimized. In addition, we address the implementation challenges of the proposed scheme. The experiments show the applicability and advantages of the proposed scheme and that it can achieve better control performance than existing switching schemes.

Contributions: The main contribution of this chapter are the following: (i) We propose a switching scheme to let a control application use both the TT and ET communication to make a trade-off between TT resource consumption and the control performance. Compared to similar existing switching schemes [56], we allow a control application to switch back to ET communication before it is settled to make more efficient utilization of the TT communication. (ii) We propose a systematic control design method that automatically designs the controllers for both the TT and ET case according to a joint performance objective while guaranteeing the switching stability. (iii) We propose a scheduling scheme that dynamically arbitrates the TT resource to optimize performance of the system at run-time. The proposed scheme is based on an offline evaluation of the switching sequences and an online scheduling algorithm that dynamically allocates the TT resource and determines the switching sequence. Furthermore, we address the implementation challenges for the switching on FlexRay protocol using a middleware solution by extending the framework in [109] onto the ET resource.

4.2 Related Works

We divide the related works in the following categories.

(i) The resource-efficient design and implementation of control applications have been studied in the recent literature under the area control/plaform co-design or control/scheduling co-design [110–115]. The main focus has been on improving either computational [110, 111, 115] or communication resource usage [113, 114]. The idea is to reduce the effective sampling frequency such that either computational or communication or both resource usage [116] are reduced. Further, there has been considerable amount of research on improving resource efficiency in the context of Networked Control System (NCS) [117–119]. Recently, there have also been many works on the co-synthesis of control and platform parameters [54, 78, 79]. There are also many works addressing the particular constraints in the automotive domain [120–125]. These related works mainly focus on the design of control and platform parameters in the static case, i.e., the schedules and the controllers do not change at run-time. In this work, we consider a case where the controllers and the communication resource allocation changes at run-time. Therefore, the results there can not be directly applied in our context.

(ii) Towards distributed embedded control system that are changing at runtime, a number of related works proposed the run-time switching scheme between the TT and ET communication with the objective of achieving decent control performance with reduced consumption of the TT resources [121, 123, 126]. This work at hand follows similar design philosophy, but differs from these works from the following aspects. These works do not address the problem of systematic control design. Usually the controllers are already given and do not provide the best control performance. Secondly, they allow only a control application to switch back to ET when the system is settled, therefore possibly using the TT communication longer than necessary. In the work at hand, we propose an automated controller design method that optimizes the joint performance and addresses the problem of switching stability. We further allow a controller to switch back to ET as early as possible with the switching stability ensured by the control design, so that more efficient utilization of the TT resource can be achieved, which in turn results in better overall control performance of the system.

4.3 Problem Formulation

4.3.1 FlexRay-based ECU Network

FlexRay-based ECU network is a commonly used architecture in the automotive setting, especially in safety-critical domains like the chassis domain. Such a system consists of a number of ECUs connected by the FlexRay bus. An ECU is a processing unit in the automotive context and has a host micro-controller running the real-time operating system that manages the resources and the execution of tasks according to their schedules. Distributed applications are partitioned into software components with data dependency that are mapped onto different ECUs and the data between them are transmitted over the FlexRay network.

The FlexRay communication protocol is organized as a sequence of *communication cycles* of equal length. In FlexRay 2.1 [8], the series of communication cycles is repeated every 64 cycles and the *cycle counter* counts from 0 to 63. Each cycle consists of a *static segment* (ST) and a *dynamic segment* (DYN).[1] The communication paradigm in the static segment is based on the Time Division Multiple Access (TDMA) scheme. The whole segment is partitioned into *static slots* of equal length, which we denote as Δ. The static slots are indexed as $S = \{1, ..., S_{ST}\}$, where S_{ST} is the largest static slot number. If a message is assigned to a specific slot, the transmission of the message takes place exactly within the static slot. Consequently, the static segment provides a deterministic communication

[1]There are also a *Symbol Window* and a *Network Idle Time* (NIT), which are not used for transmitting payload data.

schedule, which might be exploited to realize constant and optimized timing properties for the distributed application if the tasks are also properly synchronized. The dynamic segment of FlexRay is subdivided into equal-lengthed *mini-slots* of duration δ. Each mini-slot can be indexed by a mini-slot counter starting from 1 up to S_{DYN}. Similar to the static segment, the communication slots that indicate time windows for admissible message transmissions are labeled with slot numbers $S = \{S_{ST} + 1, ..., S_{ST} + S_{DYN}\}$ as depicted in Figure 4.1. However, the communication scheme in the dynamic segment follows a Flexible TDMA (FTDMA) policy. In this case, if a message is sent in a cycle, it will consume a slot S_i of duration $k \cdot \delta$. If a message is not transmitted, only one mini-slot of length δ is consumed. In contrast to the static segment, where each communication slot is of fixed length Δ, the length of the dynamic slots depends on the size of the messages being transmitted. Hence, the messages transmitted in the dynamic segment may experience variable delays depending on the workload of the messages transmitted in lower slot numbers. Further, if there are not enough mini-slots available to transmit m_i in its assigned slot S_i the message gets *displaced* and is transmitted in the next available cycle. A FlexRay schedule can be denoted as $\Theta = (S_i, B_i, R_i)$, where S_i,B_i,R_i denote respectively the *communication slot*, the *base cycle* and the *repetition rate*. $S_i \in \{1, ..., S_{ST}\}$ denotes a slot in the static segment, and $S_i \in \{S_{ST} + 1, ..., S_{ST} + S_{DYN}\}$ is a slot in the dynamic segment. The repetition rate specifies the number of cycles that elapse between two feasible message transmissions and takes the value $R_i = \{2^r \mid r \in \mathbb{N}_0, r \leq 6\}$. The base cycle denotes the first cycle a message is allowed to be transmitted within 64 cycles and is constrained by $B_i < R_i$. Figure 4.1 shows an example of FlexRay schedules.

4.3.2 Distributed Feedback Control Systems

Distributed control applications: A feedback control loop implemented on a distributed embedded platform can be partitioned into three software tasks. (i) The *sensor task* senses the measurable states of the system. (ii) The *controller task* computes the control input based on the sensor measurements and the control gains. (iii) The control input is then applied to the physical system by the *actuator task.* These tasks can be mapped on different processing units and the data between them is transmitted over the communication bus. In this work, we denote a feedback control applications as C_i and the sensor, controller and actuator task as $\tau_{i,s}$, $\tau_{i,c}$ and $\tau_{i,a}$ respectively. We further consider that the sensor and controller tasks are mapped on the same processing unit and the control input is sent through message m_i on the FlexRay bus, as shown in Figure 4.2.

TT and ET case: The tasks are triggered periodically and the time between two instances of the sensor task can be denoted as the *sampling period h*. In this chapter, we consider a time-triggered operating system on the processing units. The time difference between the begin of the sensor task and the end of the actuator task represents the *sensor-to-actuator delay d*. This delay depends strongly on the interplay between the task and

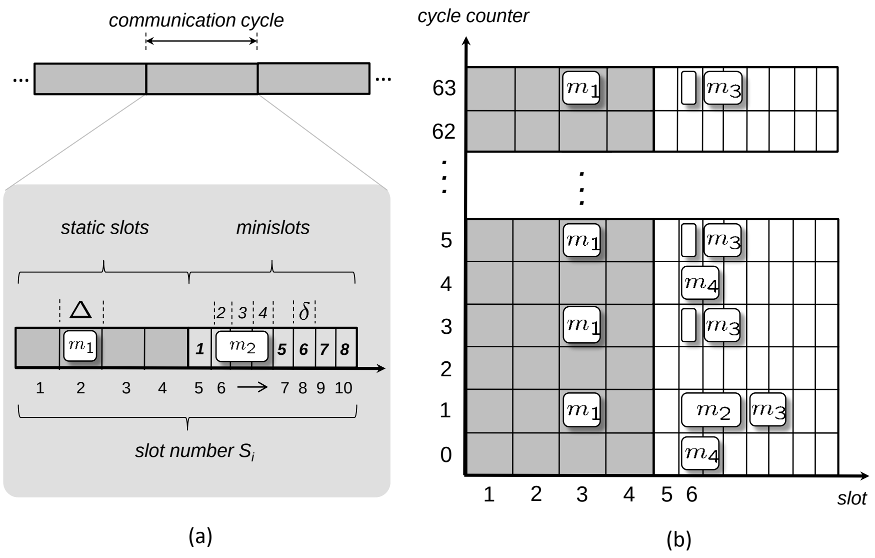

Figure 4.1: FlexRay communication: (a) The structure of communication cycles, static segment and dynamic segment. (b) An example of different messages mapped on both the static and dynamic segment.

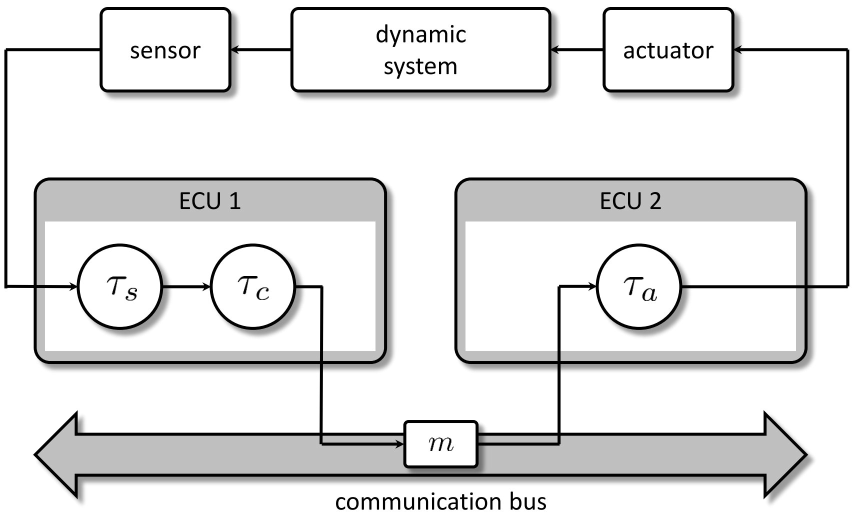

Figure 4.2: A distributed control application.

message schedules. If the tasks and messages are properly synchronized and optimized, this delay can be very small. Otherwise, it might lead to much larger delay. As already mentioned above, if the message m_i is transmitted on the static segment, the transmission time of the message is deterministic. Therefore, it is easy to design schedule to achieve very low sensor-to-actuator delay. On the other hand, if the message m_i is mapped on the dynamic segment, the transmission time depends on the run-time conditions of all the

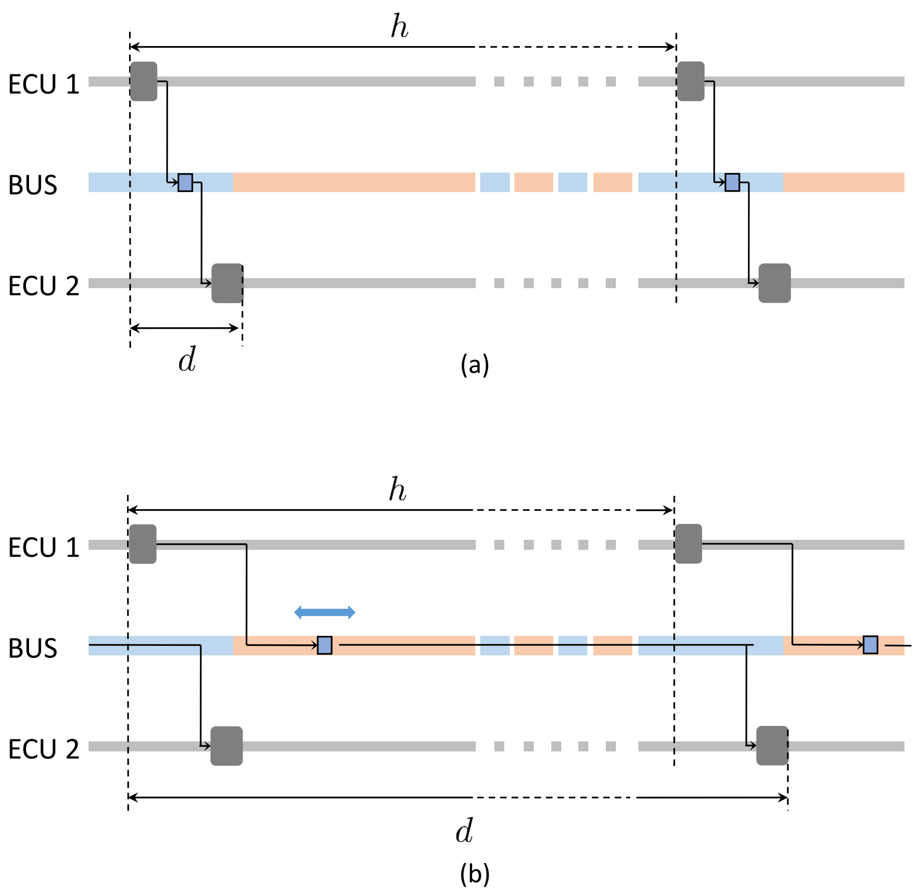

Figure 4.3: The sensor-to-actuator delay in both the case of TT communication and ET communication.

messages which have higher priorities. Therefore, it is difficult to obtain synchronized task and message schedules, thus leading to much larger delay, close to the sampling period. Figure 4.3 illustrate these two cases. Here we consider that the task schedules are properly synchronized to the TT slot schedules so that very low sensor-to-actuator delay can be achieved, i.e. $d \ll h$. On the other hand, if the message is transmitted on the dynamic segment, the system experiences a sensor-to-actuator delay of approximately one sampling period, i.e., $d \approx h$.

Feedback control system: We consider LMI systems in the context of this chapter. The continuous-time dynamics of such a system can be represented as

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t), \end{aligned} \tag{4.1}$$

where $x(t)$, $y(t)$ and $u(t)$ denote the system states, the system output and control input at time t and A, B, C denote respectively the system, input and output matrix. In the embedded implementation, the system needs to be sampled at a sampling period h and experiences a sensor-to-actuator delay d. The discretized system dynamics can be represented as

$$\begin{aligned} x[k+1] &= \Phi x[k] + \Gamma u[k] \\ y[k+1] &= Cx[k] \end{aligned} \tag{4.2}$$

where

$$\Phi = e^{Ah}, \quad \Gamma = \int_0^h e^{As} B ds \tag{4.3}$$

Here we consider two cases. For the TT case, the sensor-to-actuator delay is very small, i.e., $\tau \ll h$, the state feedback controller can be designed as

$$u[k] = K_t x[k] + F_t r. \tag{4.4}$$

For the ET case, where $\tau \approx h$, the state feedback controller can be represented as [127]

$$u[k] = K_e x[k-1] + F_e r. \tag{4.5}$$

where K_t,F_t,K_e,F_e represent respectively the feedback and static feedforware gain for the TT and ET case.

Controller design: We use pole placement technique for the controller design. For a system of n states, n poles can be placed when designing the controller. The discrete-time system requires stable poles to lie within the unit circle. The design of the control gains K_t in TT case is quite straight forward with Ackermann's formula

$$K = -\begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix} \gamma^{-1} H(\Phi), \tag{4.6}$$

where γ is the controllability matrix and

$$\begin{aligned} \gamma &= \begin{bmatrix} \Gamma & \Phi\Gamma & \cdots & \Phi^{n-1}\Gamma \end{bmatrix} \\ H(\Phi) &= (\Phi - p_1\mathbf{I})(\Phi - p_2\mathbf{I}) \cdots (\Phi - p_n\mathbf{I}). \end{aligned}$$

Here $p = \begin{bmatrix} p_1 & p_2 & \cdots & p_n \end{bmatrix}$ represent the closed-loop poles. The static feedforward gain can be computed using

$$F = \frac{1}{C(\mathbf{I} - \Phi - \Gamma K)^{-1}\Gamma}. \tag{4.7}$$

The design of controllers for the one sample delay case is more complicated. Towards this, [127] has proposed a method for pole placement. This method first transforms the system into the controllable canonical form as

$$z[k] = Tx[k], x[k] = T^{-1}z[k] \tag{4.8}$$

where T is a nonsingular transformation matrix. The system in controllable canonical form can be represented as

$$z[k+1] = \Phi_c z[k] + \Gamma_c u[k] \tag{4.9}$$

$$Phi_c = \begin{bmatrix} 0 & 1 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ a_1 & \cdots & a_{n-1} & a_n \end{bmatrix}, \quad \Gamma_c = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \tag{4.10}$$

The control input $u[k]$ can then be transformed into

$$\begin{aligned} u[k] &= Kx[k-1] = \hat{K}z[k-1] \\ &= \hat{K}_1 z_1[k-1] + \hat{K}_2 z_2[k-1] + \cdots + \hat{K}_n z_n[k-1] \\ &= \hat{K}_1 z_1[k-1] + \hat{K}_2 z_1[k] + \cdots + \hat{K}_n z_{n-1}[k] \end{aligned} \tag{4.11}$$

where $K = \hat{K}T$. By introducing a new state $z_0[k] = z_1[k-1]$, the system states can be augmented to $Z[k] = [z_0[k], z[k]]^T$ and the closed-loop augmented system becomes a $n+1$ system as

$$\begin{aligned} Z[k+1] &= \Phi_{cl} Z[k] \\ Phi_{cl} &= \begin{bmatrix} 0 & \cdots & 0 & 0 \\ 0 & \cdots & 0 & 0 \\ 0 & \cdots & 1 & 0 \\ \cdots & \cdots & \cdots & \cdots \\ \hat{K}_1 & \cdots & (a_{n-1} + \hat{K}_n) & a_n \end{bmatrix} \end{aligned} \tag{4.12}$$

where the characteristic equation can be obtained by

$$\lambda^{n+1} - a_n\lambda^n - \cdots - (a_1 + \hat{K}_2)\lambda - \hat{K}_1 = 0 \tag{4.13}$$

In comparison, where a system are placed with $n+1$ poles $p = [-p_1, -p_2, \cdots - p_{n+1}]$, the characteristic equation is

$$\begin{aligned}(\lambda+p_1)(\lambda+p_2)(\lambda+p_3)\cdots(\lambda+p_{n+1}) &= 0\\ \lambda^{n+1}+f_1(p)\lambda^n+f_2\lambda^{n-1}\cdots+f_n(p) &= 0\end{aligned} \tag{4.14}$$

By comparing the coefficients, the corresponding gain values $\hat{K}_n$ can be obtained and the original gains K can be computed by $K=\hat{K}T$. One further condition here is that $f_1(p)=\sum_1^{n+1} p_i$. Therefore the system is only stabilizable if

$$\sum_{i=1}^{n+1} p_i = -a_n \tag{4.15}$$

Therefore, the condition of the stabilizable system is

$$|a_n| < (n+1). \tag{4.16}$$

That is, if this condition is not fullfilled, there does not exist a pole set that can stabilize the system. In addition, intuitively, we can also observe that the pole placement for the one-sample delay case is more constrained than that of the zero delay case. In this chapter, these two methods will be used for the design of the TT and ET case respectively.

Performance metric and disturbance model: To measure the performance of a control system, there could be different metrics, including cost function, settling time, overshoot, etc. In the context of this chapter, without the loss of generality, we consider the primary goal of the control system is disturbance rejection. Therefore, we use the *settling time* to measure the control performance. The settling time ξ of a controller denotes the time need to reach and remain within 1% of the reference value [54]. The smaller the settling time is, the faster the control system can reject a disturbance, and thus the better the control performance is. We consider the case that the disturbance to a control system comes at the beginning of the control sample and for an individual control application, a disturbance only comes when the last one has been rejected. The disturbances of different control applications can come with random offsets. Furthermore, we also take into consideration, the limit of the actuator u_{max}, which is the maximal control input that the actuator can exert on the physical system.

Stability of switched control systems: For a set of discrete-time Linear Time-Invariant (LTI) system in the form of

$$x[k+1] = \Phi_{cl,k}x[k], \tag{4.17}$$

where $\Phi_{cl,k}$ denote the closed-loop system matrix of each system in the set. The necessary and sufficient condition [128, 129] that the arbitrary switching between them is stable is that there exists a positive definite matrix P, such that the following condition is satisfied

$$\forall k, \quad \Phi_{cl,k}^{T} P \Phi_{cl,k} - P < 0. \tag{4.18}$$

$V(x) = x^T P x$ is the Common Quadratic Lyapunov Function (CQLF) of the set.

4.3.3 Motivational Example and Problem Formulation

Towards designing such a distributed control system, there are two conventional ways. Either the message is transmitted using the TT communication, where the system experiences a negligible delay. This allows the design of faster controllers so that the system can reject disturbance faster. However, it will consume valuable TT communication resources. On the other hand, the usage of ET communication lead to much larger delay (closer to h), where comparatively slower controller can be designed. Towards this trade-off, [56] has shown that a solution in between can be achieved by allowing the controller to be switched at run-time and only use the TT slots when necessary, thus allowing the TT slot to be shared between multiple control applications to enhance the performance. However, there is still room for optimization, as shown in the following motivational example.

Motivational example: We consider an example consisting of two distributed controllers C_1 and C_2. Both are a DC motor position control system with the following continuous-time dynamics

$$\begin{aligned}\dot{x}(t) &= \begin{bmatrix} 0 & 1 & 0 \\ 0 & -10 & 1 \\ 0 & -0.02 & -2 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix} u(t) \\ y(t) &= \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} x(t).\end{aligned} \tag{4.19}$$

Both controllers have a sampling period of 50 ms. The controllers are designed so that the closed-loop poles for the TT and ET case are respectively $p_t = [0.3606, 0.4224, 0.4956]$, $p_e = [0.6171, 0.6816, 0.6153, 0.5974]$. We assume that a disturbance for both controllers comes simultaneously at the beginning of the sampling period and the control goal is to reject this disturbance as fast as possible. Here we consider four different schemes: (1) Both C_1 and C_2 use the TT slot. (2) Both C_1 and C_2 use the ET slot. (3) C_1 switches to the TT slot as soon as possible and stay there for a *dwell time* defined as larger than the settling time of the TT case and makes the TT slot available for C_2, as proposed in [56]. (4) C_1 switches to the TT slot first and uses it for several samples (possibly before settled) and then switches back to ET and releases the TT slot for C_2. For both switched schemes, we only use one TT slot. Figure 4.4 and Figure 4.5 show respectively the response of C_1 and C_2 for the four schemes. In Figure 4.4, the settling time of C_1 in four cases have the following relation ship: (1) = (3) = (4) < (2). For C_2 in Figure 4.5, this relationship is (1) < (4) < (3) < (2). The details of the settling time and resource consumption are shown

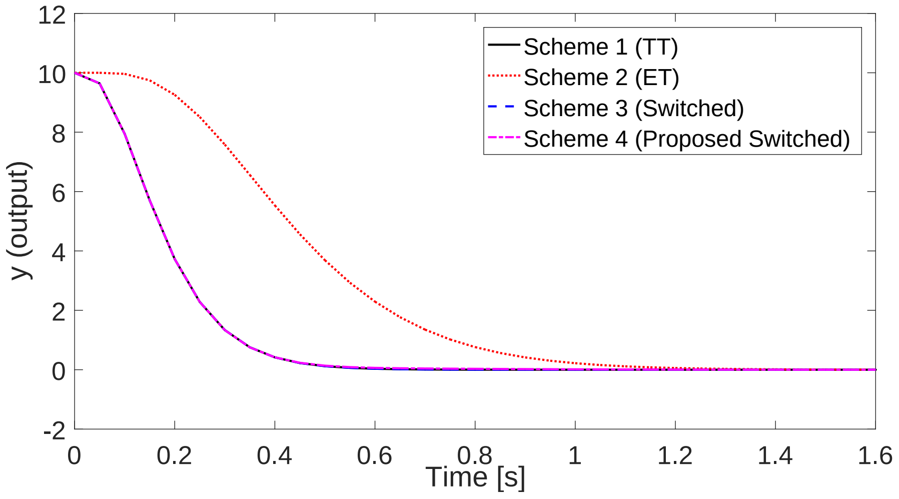

Figure 4.4: Response of C_1 for the four schemes.

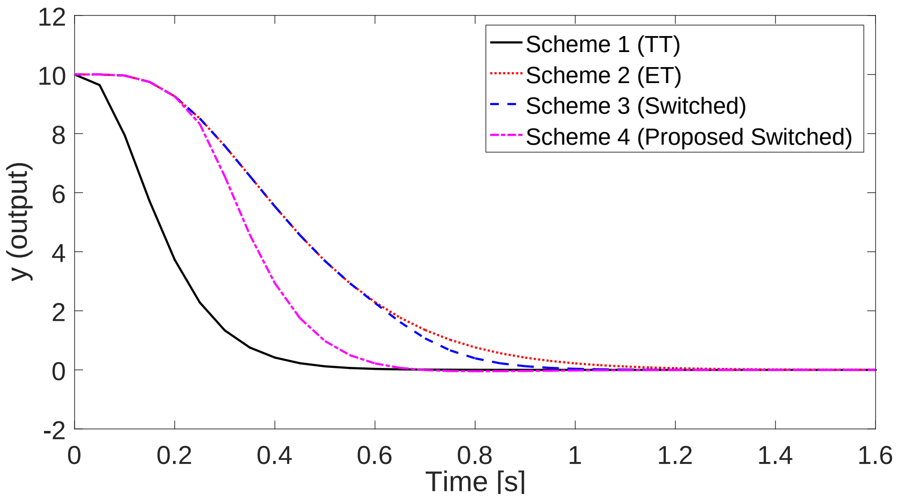

Figure 4.5: Response of C_2 for the four schemes.

in Table 4.1. As it can be observed in the results, scheme (1) has the lowest average settling time, but consumes two TT slots. Scheme (2) consumes no TT slots, but has relatively high settling times. Scheme (3) and (4) both uses one TT slot, but the average settling time of (4) is considerably lower than (3). This is due to the reason that the TT slot is released much earlier by C_1 so that it can be used by C_2 to enhance the control performance. In this work, we exploit this fact to design a switching scheme that makes more efficient utilization of the communication resources while offering better control performance.

Scheme	TT slots	$J_1[s]$	$J_2[s]$	$J_{avg}[s]$
1	2	0.55	0.55	0.55
2	0	1.15	1.15	1.15
3	1	0.55	0.95	0.75
4	1	0.55	0.65	0.60

Table 4.1: Comparison of control performance and resource utilization in the motivational example.

4.3.4 Problem Formulation

In this chapter, we consider the case that a group of M controllers where $C_i \in \mathcal{C}$ are mapped on a FlexRay based ECU network. Each controller is divided into the sensor, controller and actuator tasks denoted respectively as $\tau_{i,s}$, $\tau_{i,c}$, $\tau_{i,a}$. The sensor tasks and the controller tasks are mapped onto the same ECU and the control input m_i is transmitted over the FlexRay network. The control systems $\mathcal{C}$ have the sampling period h. N TT slots, where $N < M$, are available and each control application also has an ET slot. Each control system C_i has two subsystems: (i) $C_{i,t}$ uses the controller in Eq. (4.4) and the control gains $K_{i,t}$ and $F_{i,t}$. (ii) $C_{i,e}$ uses the controller in Eq. (4.5) and the control gains $K_{i,e}$ and $F_{i,e}$.

The switching case we consider is the following. For each control application, when a disturbance arrives, the controller firstly uses $n_{i,e}$ samples of $C_{i,e}$ and then switches to $C_{i,t}$ and uses $n_{i,t}$ samples before switching back to $C_{i,e}$ remain there until settled. Both $n_{i,e}$ and $n_{i,t}$ can take the value $n_{i,e} \geq 0$ and $n_{i,t} \geq 0$. M control applications are allowed to share the N TT communication slots and one TT slot can be occupied by only one control application simultaneously. When a control application is not using the TT slot, it uses the ET slot assigned to it.

In this work, we address the following two problems. The first problem is to design controllers $K_{i,t}$ and $F_{i,t}$ for $C_{i,t}$ and $K_{i,e}$ and $F_{i,e}$ for $C_{i,e}$ so that the switching between $C_{i,t}$ and $C_{i,e}$ is stable and the combined performance is optimized. The second problem is to design a scheme that decides the allocation of the TT slots as well as the switching sequence for each control application so that the overall control performance is optimized. This problem involves the design of $n_{i,t}$ and $n_{i,e}$ for all the controllers not yet settled.

4.4 Control Design

In this section, we introduce an automated controller design approach that designs jointly the controllers for both the TT and ET case so that the switching between these two subsystems are stable and the joint performance is optimized.

4.4.1 Switching Stability

To ensure the switching stability, we need to design the controllers for both cases in Eq. (4.4) and Eq. (4.5) so that a CQLF exists. To establish this condition, we first transform the close-loop system in both cases into the same form. Towards this, we define a new state vector $z[k] = [x[k], x[k-1]]^T$. The closed-loop system for both TT and ET case can be represented respectively as Eq. (4.20) and Eq. (4.21),

$$z[k+1] = \begin{bmatrix} \Phi + \Gamma K_t & \mathbf{0} \\ \mathbf{I} & \mathbf{0} \end{bmatrix} z[k] = \Phi_{cl,t} z[k], \tag{4.20}$$

$$z[k+1] = \begin{bmatrix} \Phi & \Gamma K_e \\ \mathbf{I} & \mathbf{0} \end{bmatrix} z[k] = \Phi_{cl,e} z[k]. \tag{4.21}$$

Therefore, if there exists a positive definite matrix P so that the LMI in Eq. (4.22) are satisfied, the switching between the two systems are stable. This condition will be used in the control design method to ensure the switching stability of the designed controllers.

$$\begin{aligned} \Phi_{cl,t}^T P \Phi_{cl,t} - P &< 0 \\ \Phi_{cl,e}^T P \Phi_{cl,e} - P &< 0 \end{aligned} \tag{4.22}$$

4.4.2 Controller Design

The goal of control design is to design the gains $K_{i,t}$, $F_{i,t}$ for $C_{i,t}$ and $K_{i,e}$, $F_{i,e}$ for $C_{i,e}$ that the control performance is optimized and the switching stability between the two cases is guaranteed. Towards this, it is impossible to provide a closed-form formulation, especially considering the LMI equations in Eq. (4.22) and the actuator limit. Therefore, we propose an controller design method based on the particle swarm optimization (PSO).

Particle swarm optimization: PSO is one common method that can be employed to solve non-convex, non-linear optimization problems and has been used for controller design in related works [107]. The basic principle of PSO is to use the evolution of the particles (or parameter set) to approach the optimal solution. Firstly, a group of particles are randomly generated and a number of iterations are carried out where in each iteration, the particles update their position. This update is guided by the *local best point* - the best point that a particle has reached - and the *global best point* - the best point that has been reached by all participles. In each iteration, the position of a particle $\sigma_{j,n}$ is computed by adding the current position $\sigma_{j,c}$ and the velocity $v_{j,n}$,

$$\sigma_{j,n} = \sigma_{j,c} + v_{j,n}. \tag{4.23}$$

This velocity is in turn computed by the following equation,

$$v_{j,n} = \alpha_0 v_{j,c} + \alpha_1 \mathcal{R}(0,1)(\sigma_{j,lb} - \sigma_{j,c}) + \alpha_2 \mathcal{R}(0,1)(\sigma_{gb} - \sigma_{j,c}), \tag{4.24}$$

where $v_{i,c}$, $\sigma_{i,b}$, σ_{gb} denote respectively the current velocity, the local best point and the global best point. α_0, α_1 and α_2 are constants that can be configured. $\mathcal{R}(0,1)$ denotes the random number generation between 0 and 1. The optimization process will finish once all the particles converge or the number of iterations is reached.

Automated design flow: The proposed design flow is illustrated in Algorithm 2. For simplicity and clarity reasons, we drop the subscript i in the algorithm description denoting the ith control application, since the controllers are design for each individual control application separately. The inputs Φ, Γ, C represent the discrete-time system matrices of the plant dynamics. h and u_{lim} denote the sampling period and the actuator limit. The outputs are the control gains K_t, F_t for C_t and K_e, F_e for C_e. Line 2 - 8 generate a number of valid particles, where each particle σ_j denotes a set of poles for both the TT and ET case. Line 4 generates randomly a pole set and Line 5 checks the validity of the set. Once the particles are initialized, they are moved for a specified number of iterations in Line 11 - 19. Line 13 moves the particle according to Eq. (4.23) and Eq. (4.24). Line 14 checks the validity of the updated pole set, designs the control gains and evaluates the performance. The local and global best points are updated throughout the iterations. Finally, the results of the global best point are returned. Algorithm 3 explains the function jointDesign. Here p in the input denotes the poles set, which is a vector of $2 \cdot nStates$, where $nStates$ is the number of the states of the system. The output J is the control performance, i.e., the settling time. Line 1 first augments the system for the ET case and also adds one additional pole to the pole set according to the control design method introduced in [127]. Recall in Section 4.3 we explained that this method requires $nStates + 1$ poles and the system is only stabilizable under certain conditions. If the system is stabilizable, Line 3 and Line 4 use pole placement techniques to design control gains for both the TT and ET case. The existence of the switching stability is checked in Line 5 using the LMI derived in Eq. (4.22). Line 7 - 8 then evaluates the performance of both cases through simulation and the joint performance is computed as a weighted sum of individual performance in Line 11. Valid control design is returned if C_e is stabilizable, a CQLF between C_t and C_e exists and the actuator limit is not exceeded.

4.5 Resource-Aware Switching Scheme

As shown in the motivational example in Section 4.3, by allowing one controller to switch back to the ET case before it is completely settled, it can release the TT resource much earlier with small trade-off in the performance. This TT resource can be utilized by other controllers to significantly increase their performance so that better overall performance of the system can be achieved. However, to obtain the optimal switching sequence is not

Algorithm 2 Joint Control Design for C_t and C_e

```
Input: $\Phi$,$\Gamma$,$C$,$h$,$u_{lim}$
Output: $K_t$,$F_t$,$K_e$,$F_e$
Parameters: $nParticles$,$nIterations$,$\alpha_0$,$\alpha_1$,$\alpha_2$
 1: $nStates$ = getNumberOfStates($\Phi$)
 2: for $j = 1$ to $nParticles$ do
 3:     while $true$ do
 4:         $\sigma_j$ = generatePoles($2 * nStates$) # generate random poles within unit circle
 5:         $[valid, K_t, F_t, K_e, F_e, J]$ = jointDesign($\sigma_j, \Phi, \Gamma, C, h, u_{lim}$) # check if result is valid
 6:         breakIFTrue($valid$)
 7:     end while
 8: end for
 9: $\{\sigma_{j,lb}\}$ = initializeLocalBest()
10: $\sigma_{gb}$ = initializeGlobalBest()
11: for $n = 1$ to $nIterations$ do
12:     for $j = 1$ to $nParticles$ do
13:         $\sigma_j$ = updateParticle($\sigma_j, \alpha_0, \alpha_1, \alpha_2$) # update the value of poles for the particle
14:         $[valid, K_t, F_t, K_e, F_e, J]$ = jointDesign($\sigma_j, \Phi, \Gamma, C, h, u_{lim}$) # check if result is valid
15:         $\sigma_{j,lb}$ = updateLocalBest($\sigma_{j,lb}, \sigma_j, J, valid$)
16:         $\sigma_{gb}$ = updateGlobalBest($\sigma_{gb}, \sigma_j, J, valid$)
17:     end for
18:     breakIfCoverged$\{\sigma_j\}$
19: end for
20: $[valid, K_t, F_t, K_e, F_e, J]$ = jointDesign($\sigma_{gb}, \Phi, \Gamma, C, h, u_{lim}$) # compute gains for the best case
21: return $K_t$,$F_t$, $K_e$, $F_e$
```

a trivial matter. In this section, we propose an online scheduling approach that decide dynamically the allocation of the TT resource and the switching of the controller so that the overall control performance of the system can be achieved.

4.5.1 The Proposed Switching Scheme

We first divide the state of a control application C_i into two general states, namely the *steady state* and the *transient state*. The transient state refers to the state between the arrival of a disturbance and time when the disturbance is rejected, i.e., the system is settled. In the proposed switching scheme, when a control application is in the steady state, ET communication is used and the controller resides in $C_{i,e}$. When a disturbance arrives, the control application enters the transient state. It will first stay in $C_{i,e}$ for $n_{i,e}$ samples, where $n_{i,e} \geq 0$. It then switch to $C_{i,t}$ and utilizes the TT communication for $n_{i,t}$ samples, where $n_{i,t} \geq 0$. The controller then switches back to $C_{i,e}$ no matter whether it is still in transient state or has returned to the steady state and stay there until it is settled. Since the control design method proposed in Section 4.4 ensures the switching stability, the system will not become unstable if it is switched back from the TT to the ET case in the transient state. The goal here is to optimize the overall control performance of a system consisting of multiple control applications by deciding on the scheduling of the utilization

Algorithm 3 Function **jointDesign** - joint design for C_t and C_e

```
Input: $\Phi$,$\Gamma$,$C$,$h$,$u_{lim}$,$p$
Output: $valid$,$K_t$,$F_t$,$K_e$,$F_e$,$J$
Parameters: $w_t$,$w_e$
Initialization: $valid = false$
 1: $[p^*, isStabilizable]$ = augmentPoles($p, \Phi, \Gamma, C$) # augment system in ET case
 2: returnIfFalse($isStabilizable$)
 3: $[K_t, F_t]$ = placePolesTT($p^*$)# pole placement for TT case
 4: $[K_e, F_e]$ = placePolesET($p^*$)# pole placement for ET case
 5: $exist$ = checkCQLF($\Phi, \Gamma, K_t, F_t, K_e, F_e$)# check if a CQLF exists
 6: returnIfFalse($exist$)
 7: $[J_t, u_{max,t}]$ = evalPerfTT($\Phi, \Gamma_i, K_t, F_t, h$)# performance evaluation for TT case
 8: $[J_e, u_{max,e}]$ = evalPerfET($\Phi, \Gamma_i, K_e, F_e, h$)# performance evaluation for ET case
 9: returnIfFalse($u_{max,t} \leq u_{lim}$ and $u_{max,t} \leq u_{lim}$)
10: $J = w_t J_t + w_e J_e$
11: $valid = true$
12: return $valid$,$K_t$,$F_t$,$K_e$,$F_e$,$J$
```

of the TT communication, which also can be translated into the problem of deciding $n_{i,e}$ and $n_{i,t}$ for each controller.

Towards this, we propose a online scheduling algorithm based on the prediction of the settling time of the controllers currently in the transient state. In a system of multiple control applications, when the disturbance for one or multiple control applications arrive, this algorithm is executed once to schedule the TT resource utilization and thus the switching sequence of the controllers so that the overall control performance is optimized. The overall control performance can either be a weighted average or the maximal value of the settling time of all controllers. The system will then follow the schedules computed until all disturbances are rejected. In the case when a new disturbance arrives, the schedules are updated.

We divide the proposed scheme into two parts. The first part is the offline evaluation of the switching sequences. For this, we generate a table containing necessary information about the relation between the switching sequence and the resulted control performance. The tables for the controllers are then stored and a scheduling algorithm is employed at run-time to predict the settling time from the tables and decide dynamically the allocation of the TT resource and the switching sequence of all the controllers. Figure 4.6 illustrates this switching scheme. In the following part of this section, we explain the approach used to generate the table through offline evaluation of the switching sequences and the online scheduling algorithm based on performance prediction.

4.5.2 Offline Evaluation of Switching Sequences

The online decision is based on the prediction of the settling time of the individual controllers, i.e., given a particular situation, what is the remaining time that is needed for the

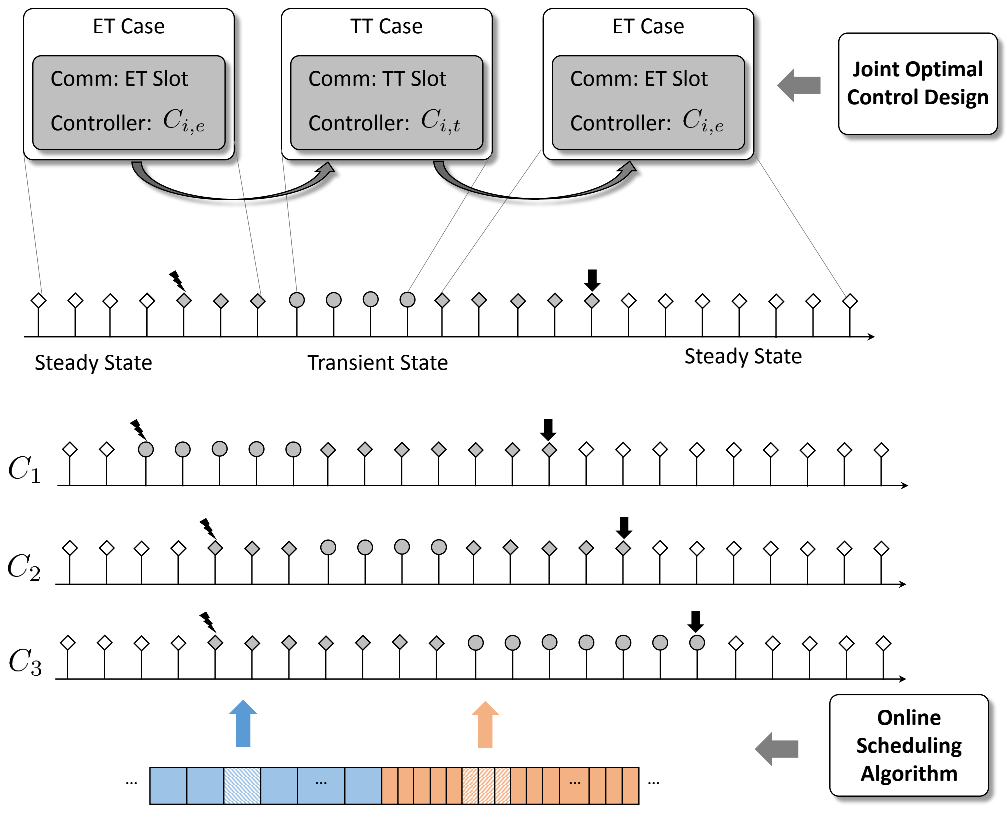

Figure 4.6: The proposed switching scheme.

system to settle. Towards this, we first compute offline the possible switching sequences of each individual controller and obtain a table containing the meaningful switching sequences and their corresponding control performance, i.e., the settling time. Given a control system C_i of sampling period h and designed control gains $K_{i,t}$ and $F_{i,t}$ for $C_{i,t}$ and $K_{i,e}$ and $F_{i,e}$ for $C_{i,e}$, the performance of the control system J_i depends directly on the number of ET samples used $n_{i,e}$ and the sample TT samples used $n_{i,t}$ before switching back to ET again. Here we would like to obtain for each control system a minimal table Λ_i containing entries $\lambda_{i,j} = [n_{i,e,j}, n_{i,t,j}, J_{i,j}] \in \Lambda_i$ that captures the relation between the switching sequence $[n_{i,e}, n_{i,t}]$ and the control performance J_i. This is done through offline simulation of the control system with the switching sequences and reducing redundant results. Algorithm 4 shows the algorithm to generate such a table.

Line 1 first evaluates the number of samples *nSamples* to reject a disturbance in the ET case. Line 2 - 8 then iterates all possible combinations of n_t and n_e provides that $n_t + n_e \leq nSamples$. The performance of each sequence is evaluated in Line 3. The switching sequence is added to the table, only when the actuator limit is not exceeded. The table obtained till Line 8 is the full table capturing all valid switching sequences. However, in order to reduce the computational complexity of the online scheduling, we further reduce the

Algorithm 4 Offline Evaluation of Switching Sequences

```
Input: $\Phi$,$\Gamma$,$C$,$h$,$K_t$,$F_t$,$K_e$,$F_e$,$u_{lim}$
Output: $\Lambda$
Initialization: $\Lambda = \emptyset$
 1: $nSamples$ = evalPerfET($\Phi, \Gamma, C, h, K_e, F_e$)$/h$
    # obtain a full table with all valid switching sequences
 2: for all $n_e$,$n_t$, where $n_e \geq 0$, $n_t \geq 0$ and $n_e + n_t \leq nSamples$ do
 3:   $[J, u_{max}]$ = evalPerfSW($\Phi, \Gamma, C, h, K_t, F_t, K_e, F_e, n_e, n_t$)
 4:   if $u_{max} \leq u_{lim}$ then
 5:     $\lambda = [n_e, n_t, J]$
 6:     $\Lambda$ = addEntry($\Lambda, \lambda$)
 7:   end if
 8: end for
    # table reduction
 9: for all $n_e, \exists \lambda_j \in \Lambda$ and $\lambda_j.n_e == n_e$ do
10:   $\Lambda_{n_e}$ = getAllSequencesWithCondition($\Lambda, n_e == \lambda_j.n_e$)
11:   $J_{n_e,min} = min(\Lambda_{n_e}.J)$
12:   if $J_{n_e,min} \geq nSamples * h$ then
13:     removeEntries($\Lambda, \Lambda_{n_e}$) # remove cases: no improvement through switching
14:   else
15:     for all $\lambda_j \in \Lambda_{n_e}$ do
16:       if $\forall \lambda_k \in \Lambda_{n_e}$ and $\lambda_k.n_t > \lambda_j.n_t$and $\lambda_k.J \geq \lambda_j.J$ then
17:         removeEntries($\Lambda, \{\lambda_k\}$) # remove cases: no improvement through more TT samples
18:       end if
19:     end for
20:   end if
21: end for
22: return $\Lambda$
```

table by removing some switching sequences. Line 9 - 21 performs this reduction. Firstly, for each possible value of n_e (Line 9), if none of the switching sequences with this value of ET samples can achieve a better performance than the non-switched case (Line 12), these sequences are not meaningful anymore. Thus we remove all these sequences (Line 13). Furthermore, for switching sequences with the same n_e value, if for a table entry λ_j, all the sequences λ_k with the same value of n_e and larger values of n_t have equal or worst performances (Line 16), these sequences $\{\lambda_k\}$ are also removed (Line 17). This is because that assigning more TT samples would not lead to better performance. This algorithm would generate a reduced table that preserves all the necessary information for the prediction of the settling time. This table can then be stored for online scheduling.

4.5.3 Online Switching based on Performance Prediction

The goal of the online switching algorithm is to determine the utilization sequence of the TT slots and thus the switching sequence of the control systems, so that the overall control performance of the whole system is optimized. This decision algorithm is run whenever a

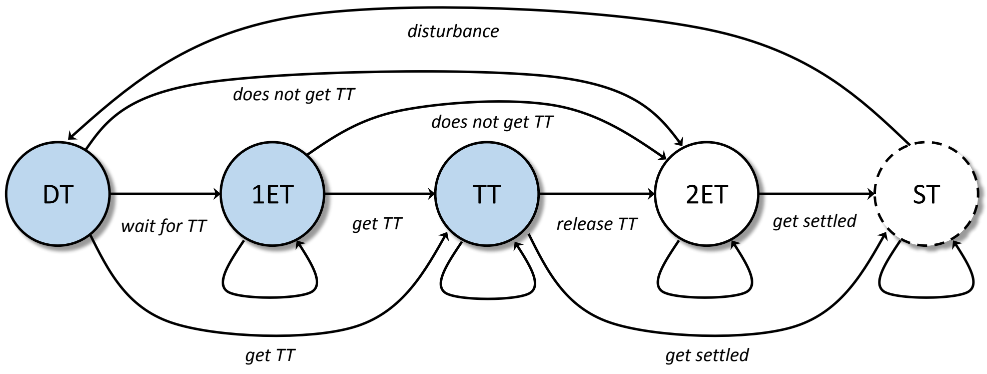

Figure 4.7: States of a controller.

disturbance arrives and updates the switching sequence of the controllers based on the past and current information.

Control application and TT slot states: Firstly, we refine the states of a control application. A control application has the following states: (i) The *disturbed state* refer the state where a disturbance just happens and no control sample has been executed yet. (ii) The *first ET state* refers to the states where the controller has used several ET samples but not yet switched to the TT samples. (iii) A control system in the *TT state* has already used several TT samples. (iv) A control system in *second ET state* has already switched back to the ET samples and will remain there until settled. (v) The settled state refers to the state where the last disturbance has been rejected and the next disturbance has not yet arrived. State (i) - (iv) correspond to the transient state (solid circle in Figure 4.7) and state (v) corresponds to the steady state (dashed circle in Figure 4.7). The state transition diagram as well as the corresponding utilization of TT and ET slots are shown in Figure 4.7. We define the state of a controller C_i as π_i. We refer the control system in state (i) to (iii) as in *active states* (blue in Figure 4.7) and consider only these controllers while deciding on the switching sequence. We further consider the initial sequence of controllers $\psi_{i,o} = (n_{i,e,o}, n_{i,t,o})$, which refers to the number of samples that the controller has already used when the algorithm is started. Non-zero initial sequence can happen when a disturbance for a control applications arrives when the disturbances of other control applications have not yet been rejected. Furthermore, for each TT slot, an variable ρ_j is defined denoting the control application that is currently using the TT slot.

Performance objective: The performance objective is to optimize the overall performance of the active control systems. This can either be configured as the average settling time or the maximal value of settling times. Without the loss of generality, we consider first the average settling time. The overall control performance J_o can be defined as $J_o = \sum w_i J_i$,

Algorithm 5 Online Scheduling

```
Input: $\mathcal{C}$,$\Pi$,$\Psi_o$,$\rho$,$\{\Lambda_i\}$,$W$
Output: $\Psi_{i,n}$,$\epsilon_b$
 1: $[\mathcal{C}_a, \Pi_a]$ = collectActiveCons($\mathcal{C}, \Pi$)
 2: $E$ = generateValidSlotMapping($\mathcal{C}_a, \Pi_a, \rho$) # generate all valid slot mapping
 3: $[\epsilon_b, \{\eta_b\}, J_b^\epsilon, \{\omega_b\}]$ = initializeBestResults()
    # iterate through all mapping
 4: for all $\epsilon_k \in E$ do
       # iterate through TT slots in a mapping
 5:    for all $\epsilon_{k,j} \in \epsilon_k$ do
 6:       $[\eta_b, J_b^\eta, \omega_b]$ = initializeBestOrderAndSchedules()
 7:       $H_{k,j}$ = generateValidOrder($\epsilon_{k,j}, \Pi, \rho$)
          # iterate through all orderings
 8:       for all $\eta_{k,j,m} \in H_{k,j}$ do
 9:          $\Omega$ = generateValidSchedules($\eta_{k,j,m}, \Psi_o$)
10:          for all $\omega_n \in \Omega$ do
11:             $J$ = evaluatePerformance($\omega_n, \Psi_{i,o}, \{\Lambda_i\}, W$)
12:             if $J < J_b^\eta$ then
13:                $[\eta_b, J_b^\eta, \omega_b]$ = updateBestOrderAndSchedules($\eta_{k,j,m}, \omega_n, J, \Psi_o$)
14:             end if
15:          end for
16:       end for
17:       $[\epsilon_k, \{\eta_b\}_k, J_k^\epsilon, \{\omega_b\}_k]$ = addToSlotMapping($\eta_b, J_b^\eta, \omega_b$)
18:    end for
19:    $[\epsilon_b, \{\eta_b\}, J_{o,b}^\epsilon, \{\omega_b\}]$ = updateBestSlotMapping($\epsilon_k, \{\eta_b\}_k, J_b^\epsilon, \{\omega_b\}_k$)
20: end for
21: $\Psi_n$ = generateFutureSchedules($\epsilon_b, \{\eta_b\}, \{\omega_b\}$) # translate results for future schedules
22: return $\Psi_n$,$\epsilon_b$
```

where J_i denotes the performance of a control applications currently in active states. Control applications already in the second ET state are not considered since they will remain with the ET slot until settled and do not need to be considered for scheduling.

Scheduling algorithm: Algorithm 5 shows the proposed online scheduling algorithm. It evaluates all valid schedules of for the TT resource allocation and returns the best one in terms of the overall control performance that is allowed by the switching scheme. The input of the algorithm contains $\mathcal{C}$ - the control applications, Π - the states of the control applications where $\pi_i \in \Pi$, Ψ_o - the old sample counter for each control application where $\psi_{i,o} \in \Psi_o$, ρ - current states of the TT slots, $\{\Lambda_i\}$ - the tables for the switching sequences for each control application and W - the weights of the control applications where $w_i \in W$. The output consists of Ψ_n - the new sample counter for the future schedules and ϵ_b - the mapping of control applications to TT slots for the future. Line 1 of the algorithm collects the control applications in active states, i.e., in disturbed state, first ET state or TT state and only these control applications will be considered in the scheduling. Line 2 then generates all possible controller to TT slot mappings with the constraint that the control applications currently in TT states remains mapped to the TT slot currently being used. Line 4 - 20 then evaluates all valid schedules. For each slot mapping ϵ_k and the corresponding set of controllers mapped to slot j (Line 4 and Line 5), the set of valid ordering (permutation) is generated (Line 7), where $\eta_{k,j,m} = (C^1, C^2, ...)$ and C^1 is the first control application in the ordering. This is also constrained by the current state of the control applications where the control application in TT state is the first in the ordering. For each of the valid ordering, we iterate though possible schedules $\omega_n \in \Omega$, where ω_n is defined as $(n_e^1, n_t^1, n_e^2, n_t^2 ...)$, where $[n_e^1, n_t^1]$ are the number of ET and TT samples for the control application that is first in the ordering $\eta_{k,j,m}$, $[n_e^2, n_t^2]$ are the second and so on. The valid schedule ω_n is constrained by the following condition,

$$\forall (C_i \in \eta_{k,j,m} \wedge C_i = C^p), \lambda_k = [n_{i,e}^p + n_{i,e,o}^p, n_{i,t}^p + n_{i,t,o}^p, J_i^p] \in \Lambda_i, \tag{4.25}$$

$$\forall (C_i, C_q \in \eta_{k,j,m} \wedge C_i = C^p \wedge C_q = C^{p+1}), n_{i,e}^p + n_{i,t}^p \leq n_{q,e}^{p+1} \vee n_{q,e}^{p+1} = n_{q,t}^{p+1} = 0, \tag{4.26}$$

$$\forall (C_i \in \eta_{k,j,m} \wedge C_i = C^1 \wedge \pi_i \text{ is in TT state}), n_{i,e}^1 = 0. \tag{4.27}$$

Eq. (4.25) enforces the condition that the combination of the ET and TT samples for C_i in the ordering $\eta_{k,j,m}$ must exist in the switching sequence table that is generated offline. This condition guarantees that the switching sequence is a valid one. Eq. (4.26) further enforced when C_i and C_q are two control application in the consecutive order, either $n_{q,e}^{p+1}$ - the number of ET samples for the following control application is larger or equal to the sum of ET $n_{i,e}^p$ and TT $n_{i,t}^p$ samples of the preceding one, or the following one has zero ET and TT samples - it is not considered to use the TT communication. This condition ensures that the TT slot can only be used by one control application simultaneously. Eq. (4.27) then enforces the condition that if the first control applications in the ordering is in a TT state, it will not be switched back to the first ET state again. Then Line 8 - 16 evaluates the performance of all the ordering and and schedules for the slot mapping ϵ_k with the slot j and Line 17 adds up the result to ϵ_k. For both the generation of the schedules and the evaluation of the performance, the old sample counter $\Psi_{i,o}$ needs to be considered since the matching with the switching sequence table needs to use the accumulated number of samples. Line 19 then updates the best slot mapping ϵ_b. Line 4 - 20 will then return the slot mapping ϵ_b with the best control performance J_k^ϵ and the corresponding orderings for each TT slot $\{\eta_b\}$ and schedules $\{\omega_b\}$. Line 21 finally generate the future schedules $\Psi_{i,n}$. This and the TT slot assignment ϵ_b will be returned.

We use an example to illustrate Algorithm 5. Consider we have four control applications C_1 to C_4 and two TT slots indexed as 1 and 2 are available. C_1 is currently in the TT state and using slot 1, while C_2 to C_4 are in the settled state. A disturbance for both C_2 and C_3 comes simultaneously and the online scheduling algorithm is triggered. From Line 1, we obtain the active control applications as $\mathcal{C}_a = \{C_1, C_2, C_3\}$. Line 2 then generates the valid control application groups with respect to the control application states. In this case, the valid groups $\epsilon_1 = \{\{C_1, C_2\}, C_3\}$, $\epsilon_2 = \{\{C_1, C_3\}, C_2\}$, $\epsilon_3 = \{C_1, \{C_2, C_3\}\}$ are generated in

Line 2. For the evaluation of performance for each group, we use ϵ_1 to illustrate this. For $\epsilon_{1,1} = \{C_1, C_2\}$, only one permutation is possible $\eta_{1,1,1} = \{C_1, C_2\}$ since C_1 is currently in TT state and has to be the first in the permutation. For $\epsilon_{1,2} = C_3$, only one permutation is possible. Line 8 to 16 then evaluates the performance of all the switching sequences possible for the permutation $\{C_1, C_2\}$ according to constraints in Eq (4.25) to Eq (4.27) and select the one with the best performance. During the evaluation of the performance, the old sample counter Ψ_o has to be considered. In this case, the old sample counter for $\psi_{1,o}$, $\psi_{2,o}$ and $\psi_{3,o}$ have the following values $n_{1,e,o} \geq 0, n_{1,t,o} \geq 0$,$n_{1,e,o} = 0, n_{1,t,o} = 0$ and $n_{1,e,o} = 0, n_{1,t,o} = 0$. Therefore $n_{1,e,o}$ and $n_{1,t,o}$ have to be considered when iterating through valid schedules or evaluating performance. Similarly, this is done for the permutation in $\epsilon_{1,2}$. The sum of the performance for both $\epsilon_{1,1}$ and $\epsilon_{1,2}$ is the best performance for ϵ_1. Then Line 21 select the best control application grouping and generate the future schedules.

We can make a trade-off between the computation effort and the possibility of achieving optimality. Towards this, the condition in Eq. (4.26) can be tightened as Eq. (4.28), i.e., once a TT slot becomes available, it will be immediately assigned to another control application that is not yet settled. This considerably reduces the computation complexity of the algorithm by reducing the number of permutations that need to be evaluated.

$$\forall (C_i, C_q \in \eta_{k,j,m} \wedge C_i = C^p \wedge C_q = C^{p+1}), n_{i,e}^{p} + n_{i,t}^{p} = n_{q,e}^{p+1} \vee n_{q,e}^{p+1} = n_{q,t}^{p+1} = 0, \quad (4.28)$$

4.5.4 Middleware-based Slot Sharing

One important challenge for the switching scheme upon FlexRay protocol, which the related approaches have not addressed, is the actual implementation of the switching between the TT and ET communication. The FlexRay protocol itself does not support run-time switching of messages between the TT and ET slots. The Flexray configuration is performed offline and the communication cannot be reconfigured in a straightforward manner unless the whole FlexRay cluster is shut down and reconfigured.

Towards addressing this problem, we propose a middleware based solution to implement the switching between two schedules by extending the framework in [109] onto the dynamic segment. This solution is illustrated in Figure 4.8. The dynamic switching between the TT and ET are realized by inserting a software layer between the application data and the communication controllers. This software layer detaches the mapping of the data to signals and dynamically decides which data goes into which signal on the software level according to a slot sharing policy, which can be computed by the online scheduling algorithm. This software layer can be implemented in a lightweight fashion by a sender slot sharing task - scheduled after the data are ready and just before the slot starts - and a receiver slot sharing task - scheduled just after the slot finishes. To allow the receiver task to know and filter the corresponding data, an ID field is inserted into the TT slot before

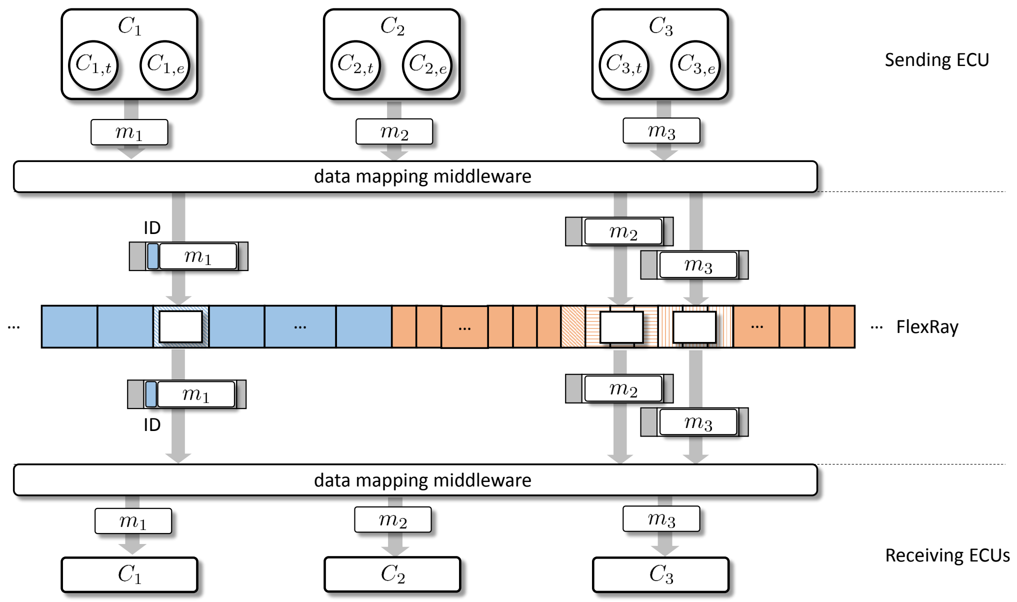

Figure 4.8: Middleware-based slot sharing.

the actual payload data, denoting the index of the control application that is currently using the TT slot. This middleware is also compatible to COTS development tools like Matlab/Simulink. One limitation of the slot sharing scheme, which is the limitation of all similar switching schemes based on FlexRay, is that the slot sharing and switching is only allowed between applications whose tasks that are sending data are mapped on the same ECU. Dynamic slot sharing between the ECUs are not possible, limited by the FlexRay protocol.

4.6 Experimental Results

In this section, we use a case study to illustrate the advantages and the applicability of the proposed switching scheme using simulation results.

4.6.1 Case Study

We consider a case study consisting of four control applications C_1 to C_4. We use two different plant models derived from common automotive applications. The first one is the DC motor position control (DCP) given in Eq. (4.19) and the second one is the cruise control system (CC) [54], whose continuous-time dynamics can be represented as

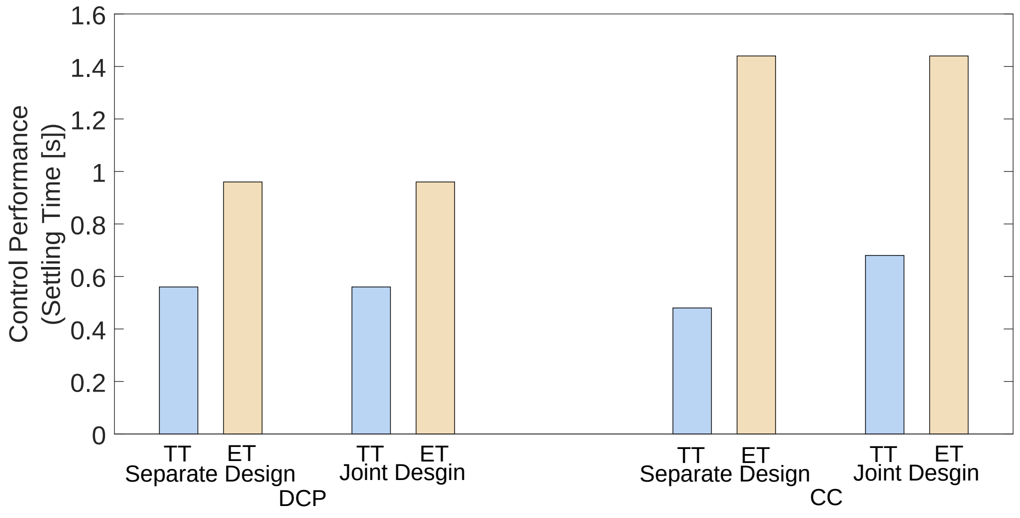

Figure 4.9: Comparison between the control performance of both TT and ET case using separate and joint design.

$$\begin{aligned}\dot{x}(t) &= \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6.0476 & -5.2856 & -0.238 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 0 \\ 2.4767 \end{bmatrix} u(t) \\ y(t) &= \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} x(t). \end{aligned} \tag{4.29}$$

We assign the plant model in Eq. (4.19) to C_1 and C_2 and the plant model in Eq. (4.29) to C_3 and C_4. All the control applications are have a sampling period of 40 ms.

4.6.2 Results and Discussions

Control design: Figure 4.9 shows the results of the controller design for both plant models. It can be observed from the figure that for the DCP system, the joint design can achieve as good control performance as the separate design. For CC, the joint design results in the same performance for the ET case, but for the TT case, the settling time is larger than the separate design. This might be due to the extra constraints for the existence of the CQLF, which render the poles in the separate design not possible.

Switching sequence and performance: We then generate the table for the switching sequences as explained in Algorithm 4 in Section 4.5. Figure 4.10 shows the comparison between different switching sequences for DCP plant model. Different lines represents the switching sequence with respectively 0, 2, 5 and 10 ET samples before switching to the TT case and the x-axis value shows the number of TT samples that are used before the controller switches back to ET. In the case of x-axis value is equal to 0, it is the same with

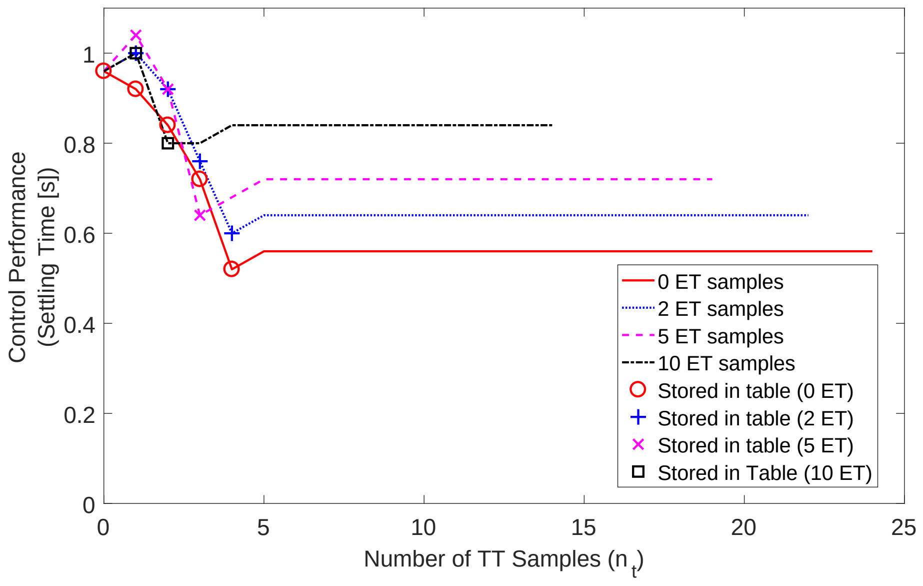

Figure 4.10: Comparison between switching sequences for DCP. Lines: ET samples before switching to TT. X-axis: TT samples before switching back to ET. Markers: entries stored in table.

pure ET case and has the same control performance. When the controller switches to TT, if the number of TT samples is small, the settling time might even increase (e.g.,blue dotted and magenta dashed line). This might be due to frequent switching. As the number of TT samples increase, the settling time shows an decreasing trend and after several samples, the settling time remains the same. This means that using extra TT resource does not increase the control performance. The markers shown in the figure denotes the meaningful switching sequences that are stored in the table for online scheduling for the set of 0, 2, 5 and 10 ET samples before switching to TT. Other sequences either exceeds the actuator limit or would waste TT resources.

Online algorithm: In order to evaluate the performance of the online scheduling algorithm, we first consider the scenario when a disturbance of all four control applications arrive at the same time. The results are obtained using the simplified algorithm with constraint in Eq. (4.28), i.e., the TT slot is assigned immediately when it becomes available. Here we consider two cases of available TT slots. For case (a) N = 1, one TT slot is available and for case (b) N = 2, two TT slots are available. Figure 4.11 and Figure 4.12 shows the control response and the utilization of the TT slots for the four applications for case (a) and (b) respectively. From Figure 4.11, we see that the scheduling algorithm assigns firstly 5 TT samples to C_4 and therefore the disturbance C_4 is rejected faster. It then assigns 3 and 2 samples of TT slots to C_2 and C_1 respectively and finally 4 TT samples to C_3. In Figure 4.12, there could be simultaneously two TT samples, and both the TT slots are assigned to C_3 and C_4 first and then to C_1, C_2. In both (a) and (b), the scheduler assigns

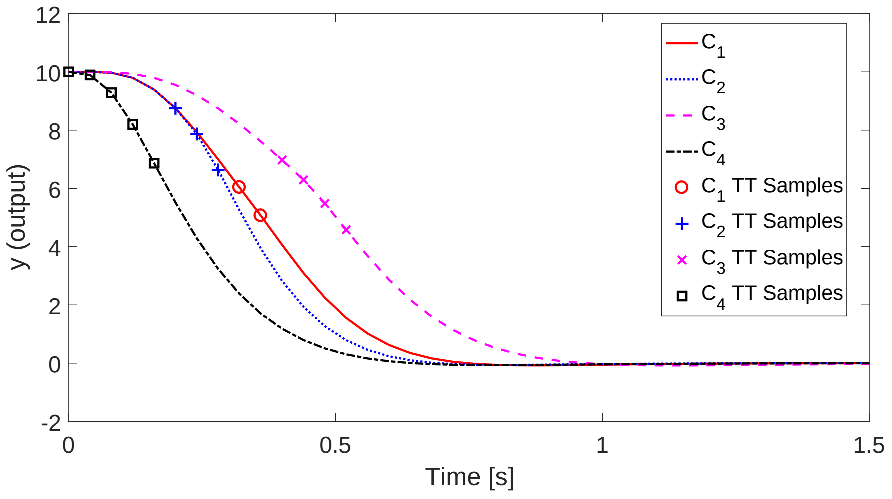

Figure 4.11: Output trajectories of the four control applications sharing one TT slot in the case of simultaneous disturbance. The markers show the TT samples.

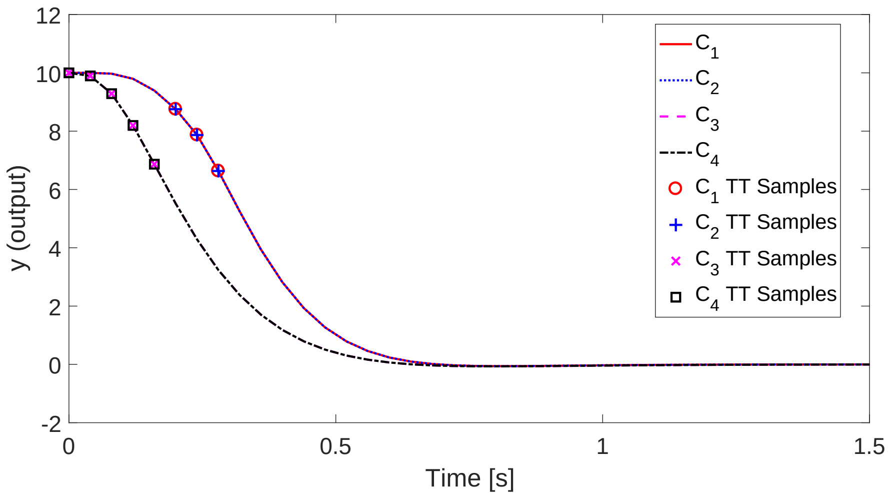

Figure 4.12: Output trajectories of the four control applications sharing two TT slots in the case of simultaneous disturbance. The markers show the TT samples.

the TT slots to C_3 and C_4 first, which are both of plant CC. Intuitively, the reason could be this plant takes longer to settle (see Figure 4.9) and assign TT slot first to them might enhance the performance more. Furthermore, Table 4.2 shows the comparison of the resource utilization and the overall control performance between (i) the purely TT case, (ii) the purely ET case, (iii) the switching scheme proposed in [56] and (iv) the switching scheme proposed in this chapter. For scheme (iii), we further consider the case of using separately design controllers and jointly designed controllers (s) that guarantees the switching stability. The results show that the control performance and the consumption of TT resources of the proposed scheme lies between the pure TT and pure ET case and close to the TT case.

1 TT slot						
Scheme	TT slots	$J_1[s]$	$J_2[s]$	$J_3[s]$	$J_4[s]$	$J_{avg}[s]$
i	4	0.56	0.56	0.48	0.48	**0.52**
ii	0	0.96	0.96	1.44	1.44	**1.20**
iii (s)	1	0.96	0.96	0.68	1.16	**0.94**
iii	1	0.96	0.96	0.48	0.92	**0.83**
iv	1	0.72	0.64	0.92	0.60	**0.72**

2 TT slots						
Scheme	TT slots	$J_1[s]$	$J_2[s]$	$J_3[s]$	$J_4[s]$	$J_{avg}[s]$
i	4	0.56	0.56	0.48	0.48	**0.52**
ii	0	0.96	0.96	1.44	1.44	**1.20**
iii (s)	2	0.92	0.92	0.68	0.68	**0.80**
iii	2	0.88	0.88	0.48	0.48	**0.68**
iv	2	0.64	0.64	0.60	0.60	**0.62**

Table 4.2: Comparison of the control performance and the resource utilization for schemes (i) - (iv).

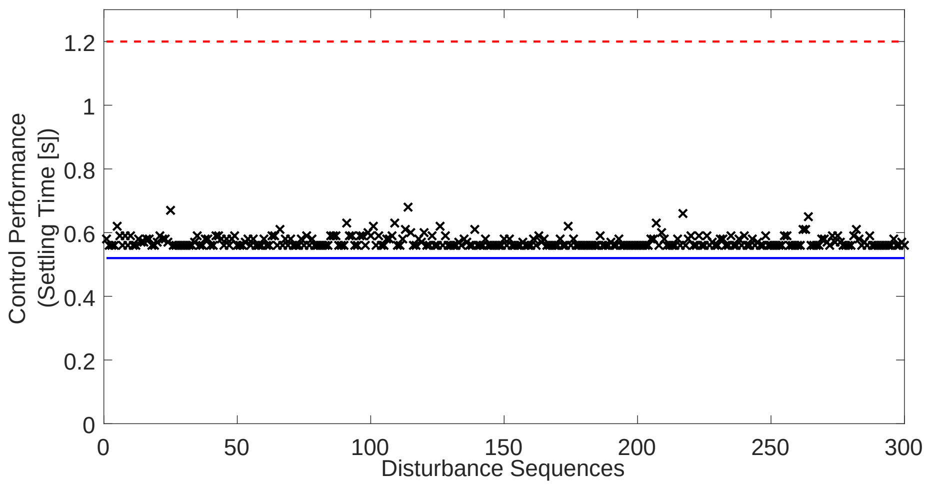

Figure 4.13: The average control performance of C_1 to C_4 with 300 randomly generated disturbance sequences. The blue solid line shows the performance of the pure TT case and the red dashed line the pure ET case.

The proposed scheme also achieves better controller performance for both cases of scheme (iii). It can also be observed that assigning more TT slots can increase the performance in both the proposed scheme and scheme (iii). What is also notable is that in the case of 2 TT slots for C_3 and C_4 in the proposed scheme, the control performance is actually little better than the pure TT case with poles from the joint design. This corresponds to the similar valley in the curves in Figure 4.10, although Figure 4.10 shows the result from DCP and C_3 and C_4 have the plant CC. We then randomly generate 300 disturbance sequences, where each disturbance sequence includes one disturbance for each control application but the arrival time between the disturbances are different. Figure 4.13 shows the average control performance using the proposed approach with 1 TT slot, compared to the case of pure TT and pure ET. The results confirm that the proposed scheme can strike a trade-off between resource utilization and the control performance and in most cases offers a control performance close to the pure TT case.

4.7 Concluding Remarks

In this chapter, we proposed a resource-aware switching scheme for distributed control applications based on hybrid communication protocols. We allow each control application to switch between TT and ET communication to make use of TT resource to enhance the control performance. Valuable TT communication resources can be shared between multiple controllers to enhance the overall performance of the system. We further allow a control application to release the TT communication resource and switch back to ET case before a disturbance is rejected to make more efficient utilization of the TT resource. Towards this, we proposed an automated design method that can design both the controllers for TT and ET case to optimize joint control performance while guaranteeing the stability of the switching. A scheduling scheme is proposed based on the offline evaluation of the switching sequences and the online scheduling algorithm to dynamically allocate the TT communication to different controllers at run-time. The experimental results have shown that the proposed scheme offers a trade-off between the TT resource consumption and the control performance and achieves better control performance compared to existing switching schemes.

Chapter 5

Concluding Remarks

The rapid development in the functionality of modern vehicles, especially in the ADAS and infotainment domain, has imposed new challenges upon the design of underlying E/E architecture. With the increasing size and complexity of the E/E architecture, meeting conventional requirements like real-time capability, resource-efficiency as well as new requirements like flexibility, adaptability and safety has become more challenging. This thesis investigates the problem of design and synthesis of automotive CPS with focus on distributed systems, where the communication network plays an important role. Design and synthesis approaches are proposed to address the requirements like real-time capability, flexibility/reconfigurability and resource-efficiency for such systems.

5.1 Summary

Firstly, one approach is introduced to co-synthesize task and communication schedules for an Ethernet-based time-triggered system. The approach targets at the time-triggered traffic in recent Ethernet protocols and synthesizes the task and communication schedules in a synchronized manner. It considers the application-level timing, where the interplay between the task and network schedules plays an important role. Compared to network timing, the application-level timing is more important for the functional performance of the distributed applications. This approach formulates important constraints for the scheduling problem and handles multi-objective optimization, where different application-level timing objectives and their combinations are considered. It further considers the switched Ethernet topology and specific parameters like the synchronization precision. The proposed approach is independent of task and communication configurations as well as network topologies and device performance. An industrial-sized case study is used to show the applicability of the method towards optimization of complex timing objectives and the experimental results also show that it can be scaled to system of reasonably large size.

Secondly, a schedule management framework is proposed to generate and manage online time-triggered schedules for the case of PnP and software updates. The framework employs a client-server architecture and utilizes both the computation and storage resource onboard the vehicle and in backend servers through the cloud-computing fashion. When a request for the re-computation of the schedules arrives, the framework can synthesize the schedules on an embedded platform as well as send the request through the cloud to the server to be synthesized. It will then select the fastest results and thus reduce the time for obtaining schedules, paving the way for future cloud-based PnP of software functions. In the synthesis process, it implements a four-staged strategy where the synthesis scales from incremental design to a complete re-synthesis, thus offering trade-off between the synthesis time and disturbance to existing applications on one side and the chance of accommodating new applications on the other. A configuration pool is proposed to facilitate the reuse of schedule configurations and thus reduce the synthesis overhead. A prototype of this software framework is developed using a Raspberry Pi as the client and a notebook computer as the server. The results show that the proposed framework can efficiently generate and manage schedules online, synthesize schedules in a time meaningful for PnP and use the staged synthesis strategy to offer trade-off between synthesis time, disturbance to existing applications and the chance of accommodating new applications.

Finally, a resource-efficient scheme is proposed for CPS based on hybrid communication protocols. This scheme allows a control application to switch between the TT and ET communication to conserve valuable TT communication resource. Compared to existing switching schemes in this setting, the approach proposed in this thesis allows a control application to use as much TT resource as meaningful and switch back to ET resource before the system has settled. This allows the TT resource to be released as early as possible so that it can be utilized by other control applications to enhance the overall control performance. Towards this, a PSO based control design method is proposed to automatically design controllers for the TT and ET case in a joint way and guarantee the switching stability. For the computation of the TT resource allocation, an online scheduling algorithm is proposed to dynamically compute the allocation of the TT resource and the switching sequence of the control applications so that the overall control performance is optimized. This online algorithm is based on the prediction of the remaining settling time, which can be drawn from a switching sequence table generated offline. The results have shown that the proposed scheme can strike a balance between TT resource utilization and the control performance and achieves better control performance compared to existing switching schemes with the same resource utilization.

5.2 Future Works

The approaches proposed in this thesis can be further extended in different aspects. In this section, the possible extensions and future works are discussed.

Extensibility driven schedule synthesis: The approach explained in Chapter 2 addresses mainly the real-time requirements. This work can further be extended to the requirements on the extensibility of the schedules. The extensibility-driven schedule synthesis is an important research direction. The motivation comes from the fact that in the automotive domain, the systems are designed in an iterative manner. In each iteration, new software applications and possibly ECUs are added to the system. During this process, the schedules of the existing applications are kept as unchanged as possible. Especially for safety-critical applications, the configurations and parameters need rigorous testing and validation. If the schedules are changed during a design iteration, the existing applications need to be tested and validated again, which introduces a lot of overhead and costs. Therefore, it would be advantageous, if the schedules are designed to be extensible in the first place. That is, more future schedules can be added to this schedule set without changing the existing schedules. In the context of TTP and FlexRay, there have already been some works on extensible scheduling [70, 72]. However, the communication systems considered there are of bus topology. The problem becomes much more challenging, when it comes to switched Ethernet, which is a point-to-point network that can take different network topologies. There are some works on extensible scheduling for time-triggered Ethernet [89]. But the intention there is to make the schedules porous for the best-effort traffic and the method is also relatively simple. To synthesize time-triggered Ethernet schedules that are extensible to future time-triggered traffic is still an open problem. The main challenge here is the much higher complexity compared to the protocols of bus topologies and the handling of the interplay between different links and routes.

Co-synthesis of mixed-criticality networks: Another possible future work in line with Chapter 2 is the co-synthesis of mixed criticality networks. Conventional automotive bus systems are tailored to the specific requirements of the domain. However, since the automotive E/E system has become increasingly of a mixed criticality nature, communication protocols offering mixed-criticality traffics have become popular. FlexRay, for example, is a mixed-criticality bus offering both TT and ET traffics. The recent Ethernet protocols also offer different classes of traffics, including scheduled traffic, rate-constraint traffic and best effort traffic. Traditionally, the schedule synthesis mainly refers to the time-triggered scheduling scheme. However, it is also possible to synthesize the configuration parameters for ET traffics. For example, priorities of CAN messages as well as Ethernet frames can also be synthesized. For these traffics, it is difficult to find a closed form solution for the problem. Rather, heuristic or meta-heuristic methods can be used in combination with timing analysis techniques to synthesize the parameters. An even more challenging yet meaningful problem is to co-synthesize the time-triggered as well as event-triggered traffics together, since in the case of Ethernet, the scheduling of the time-triggered traffic influences strongly the event-triggered traffic. The co-design of the mixed-criticality communication can evaluate the trade-off between the bandwidth assignment to the different classes of traffics and thus making more efficient utilization of the available bandwidth. This might also offer a basis for the abstraction of the communication network, which in turn contributes to the modularization of the software and the flexibility and adaptivity of the platform.

Communication abstraction: The approach introduced in Chapter 3 as well as partially in Chapter 4 contributes to the reconfigurability of the platform in that it allows the reallocation of the communication resources. Compared to the current configuration of the communication in automotive E/E architecture, where the data-to-message, message-to-schedule mapping is statically configured, the proposed approaches allow to a certain extend, the underlying communication resources to be re-distributed. An extension in this direction is the abstraction of the communication resources from the application software, where it is not necessary for the application software to be aware of the actual communication resources that transmit the data. On the other hand, the requirements for the data are specified on the application level and middleware solutions are employed to dynamically assign communication resources to the application data. In the same direction, there have been some middleware solutions that abstract the communication, e.g., SOME/IP and DDS. However, there are still many challenges to be overcome, including meeting the real-time requirements and efficient utilization of the bandwidth. Towards these, schedule synthesis and timing analysis techniques need to be properly integrated into the middleware solutions.

Parallelization of synthesis approaches: One future direction for the approach explained in Chapter 3 is the parallelization of the synthesis approaches. A four-staged strategy for the schedule synthesis is employed in this thesis, which can make a trade-off between the objectives of reducing the synthesis time and the disturbance to existing applications and increasing the chance of accommodating new applications. However, the four steps are still implemented in a serial manner. Modern processors are increasingly equipped with multiple cores. If the multi-core architecture can be utilized to parallelize the different schedule synthesis approaches, an even reduced synthesis time can be achieved, thus making the PnP of software applications in a cloud-based setting event more convenient. The main challenges to be addressed here are the implementation challenges.

Control/platform co-design: The approach proposed in Chapter 4 falls broadly into the category of control/platform co-design or co-synthesis. Although there have been many recent works addressing this problem in the CPS domain, this field is still a relatively open field with a lot of opportunities to explore. The main goal here is to consider the design problem of controller on the application level and of platform on the system level jointly to remove the design conservativeness and thus achieving better performance or reduced resource utilization. In the direction of the approach explained in Chapter 4, there are several possibilities of extension. Firstly, sample drops can be added to the TT and ET case considered in this thesis to obtain a more comprehensive situation. Secondly, event-triggered and self-triggered control can also be included into the picture, where a control application does not use any communication resource if it is not needed, thus achieving more resource savings, also for the ET communication. Thirdly, towards online prediction of control performance, machine learning techniques can possibly be utilized to train regression models or decision trees that requires less computational effort online and can cope with different initial states. As already mentioned, there are a lot of opportunities to explore towards safe, resource-efficient design of CPS. The main challenges to overcome here is the complexity problem and the heterogeneity of the modeling and design methods for control and embedded systems.

5.3 Outlook

As the technology in automotive domain advances, connected self-driving vehicles will become a reality in a not so distant future. As the functionality of modern cars becomes increasingly more powerful, the underlying E/E architecture as well as the design methods will certainly evolve. Towards this, emerging trends like ECU consolidation, Ethernet-based communication architecture, adaptive platform, connectivity and CPS-oriented design are likely to continue and gather momentum. This thesis provides several solutions in the design/synthesis of automotive CPS addressing several requirements that either arise from or are strengthened by these developing trends. Therefore, these solutions are relevant and might serve as the basis for further work towards designing future automotive E/E systems.

Bibliography

[1] M. Broy, I. H. Kruger, A. Pretschner, and C. Salzmann, “Engineering automotive software,” *Proceedings of the IEEE*, vol. 95, no. 2, pp. 356–373, 2007.

[2] R. N. Charette, “This car runs on code,” *IEEE spectrum*, vol. 46, no. 3, p. 3, 2009.

[3] T. Streichert and M. Traub, *Elektrik/Elektronik-Architekturen im Kraftfahrzeug: Modellierung und Bewertung von Echtzeitsystemen.* Springer-Verlag, 2012.

[4] C. Buckl, B. Schätz, M. Fehling, K. Kuhn, C. Klein *et al.*, “Mehr software (im) wagen: Informations-und kommunikationstechnik (ikt) als motor der elektromobilität der zukunft,” *Abschlussbericht des BMBF-geförderten Verbundvorhabens “eCar-IKT-Systemarchitektur für Elektromobilität*, 2011.

[5] F. Simonot-Lion, “In car embedded electronic architectures: How to ensure their safety,” *IFAC Proceedings Volumes*, vol. 36, no. 13, pp. 1–8, 2003.

[6] N. Navet, Y. Song, F. Simonot-Lion, and C. Wilwert, “Trends in automotive communication systems,” *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1204–1223, 2005.

[7] B. Group *et al.*, *Der neue BMW 7er: Entwicklung und Technik.* Springer-Verlag, 2009.

[8] F. Consortium *et al.*, “Flexray communications system protocol specification version 2.1,” 2005.

[9] ——, “Flexray communication systems protocol specification, version 3.0.1,” 2010.

[10] M. Rausch, *FlexRay: Grundlagen, Funktionsweise, Anwendung.* Hanser Verlag, 2008.

[11] W. Zimmermann and R. Schmidgall, *Bussysteme in der fahrzeugtechnik: protokolle, standards und softwarearchitektur.* Springer-Verlag, 2014.

[12] C. Spurgeon, *Ethernet: the definitive guide.* " O'Reilly Media, Inc.", 2000.

[13] R. Seifert and J. Edwards, *The All-New Switch Book: The Complete Guide to LAN Switching Technology.* John Wiley & Sons, 2008.

[14] "Ieee standard for local and metropolitan area networks–media access control (mac) bridges and virtual bridged local area networks," *IEEE Std 802.1Q-2011 (Revision of IEEE Std 802.1Q-2005)*, pp. 1–1365, Aug 2011.

[15] "Time-sensitve networking task group," http://www.ieee802.org/1/pages/tsn.html, accessed: 2017-06-28.

[16] "Ieee standard for local and metropolitan area networks – bridges and bridged networks - amendment 25: Enhancements for scheduled traffic," *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q— as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q—/Cor 1-2015)*, pp. 1–57, March 2016.

[17] "Ieee standard for local and metropolitan area networks – bridges and bridged networks – amendment 26: Frame preemption," *IEEE Std 802.1Qbu-2016 (Amendment to IEEE Std 802.1Q-2014)*, pp. 1–52, Aug 2016.

[18] R. Pigan and M. Metter, *Automating with PROFINET: Industrial communication based on Industrial Ethernet.* John Wiley & Sons, 2008.

[19] D. Jansen and H. Buttner, "Real-time ethernet: the ethercat solution," *Computing and Control Engineering*, vol. 15, no. 1, pp. 16–21, 2004.

[20] A. D. N. Part, "7: Avionics full duplex switched ethernet (afdx) network," *ARINC Specification 664p7*, vol. 7, 2005.

[21] S. AS6802, "Time-triggered ethernet," *SAE International*, 2011.

[22] L. L. Bello, "The case for ethernet in automotive communications," *ACM SIGBED Review*, vol. 8, no. 4, pp. 7–15, 2011.

[23] ——, "Novel trends in automotive networks: A perspective on ethernet and the ieee audio video bridging," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, Sept 2014, pp. 1–8.

[24] S. Sommer, A. Camek, K. Becker, C. Buckl, A. Zirkler, L. Fiege, M. Armbruster, G. Spiegelberg, and A. Knoll, "Race: A centralized platform computer based architecture for automotive applications," in *Electric Vehicle Conference (IEVC), 2013 IEEE International.* IEEE, 2013, pp. 1–6.

[25] "Open alliance sig," http://www.opensig.org, accessed: 2017-06-28.

[26] F. Sagstetter, "Schedule synthesis for time-triggered automotive architectures," Ph.D. dissertation, Universitätsbibliothek der TU München, 2016.

[27] "Road vehicles – controller area network (can) – part 2: High-speed medium access unit," *ISO 11898-2:2016*, pp. 1–30, Jun 2016.

[28] "Road vehicles – controller area network (can) – part 3: Low-speed, fault-tolerant, medium-dependent interface," *ISO 11898-3:2006*, pp. 1–25, Jun 2006.

[29] F. Hartwich *et al.*, "Can with flexible data-rate," in *Proc. iCC*, 2012, pp. 1–9.

[30] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller area network (can) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, 2007.

[31] "Road vehicles – controller area network (can) – part 4: Time-triggered communication," *ISO 11898-4:2004*, pp. 1–32, Aug 2008.

[32] K. Borgeest, *Elektronik in der Fahrzeugtechnik.* Springer, 2010.

[33] L. Völker, "Some/ip-die middleware für ethernetbasierte kommunikation," *Hanser automotive networks*, 2013.

[34] G. Pardo-Castellote, "Omg data-distribution service: Architectural overview," in *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on.* IEEE, 2003, pp. 200–206.

[35] M. Wagner, S. Schildt, and M. Poehnl, "Service-oriented communication for controller area networks," in *Vehicular Technology Conference (VTC-Fall), 2016 IEEE 84th.* IEEE, 2016, pp. 1–5.

[36] AUTOSAR, "Virtual functional bus autosar cp release 4.3.0."

[37] "Road vehicles – open interface for embedded automotive applications – part 1: General structure and terms, definitions and abbreviated terms," *ISO 17356-1:2005*, pp. 1–21, Jan 2005.

[38] "Road vehicles – open interface for embedded automotive applications – part 3: Osek/vdx operating system (os)," *ISO 17356-3:2005*, pp. 1–61, Nov 2005.

[39] "Autosar," https://www.autosar.org, accessed: 2017-06-28.

[40] M. Di Natale and A. L. Sangiovanni-Vincentelli, "Moving from federated to integrated architectures in automotive: The role of standards, methods and tools," *Proceedings of the IEEE*, vol. 98, no. 4, pp. 603–620, 2010.

[41] S. Chakraborty, M. Lukasiewycz, C. Buckl, S. Fahmy, N. Chang, S. Park, Y. Kim, P. Leteinturier, and H. Adlkofer, “Embedded systems and software challenges in electric vehicles,” in *Proceedings of the conference on design, automation and test in Europe.* EDA Consortium, 2012, pp. 424–429.

[42] K. Becker, J. Frtunikj, M. Felser, L. Fiege, C. Buckl, S. Rothbauer, L. Zhang, and C. Klein, “Race rte: a runtime environment for robust fault-tolerant vehicle functions,” in *CARS 2015-Critical Automotive applications: Robustness & Safety*, 2015.

[43] J. Wei, J. M. Snider, J. Kim, J. M. Dolan, R. Rajkumar, and B. Litkouhi, “Towards a viable autonomous driving research platform,” in *Intelligent Vehicles Symposium (IV), 2013 IEEE.* IEEE, 2013, pp. 763–770.

[44] E. Ackerman, “Tesla model s: Summer software update will enable autonomous driving,” *IEEE Spectrum Cars That Think*, 2016.

[45] R. Bradley, “Tesla autopilot, the electric-vehicle maker sent its cars a software update that suddenly made autonomous driving a reality,” 2016.

[46] H.-T. Lim, B. Krebs, L. Volker, and P. Zahrer, “Performance evaluation of the inter-domain communication in a switched ethernet based in-car network,” in *Local Computer Networks (LCN), 2011 IEEE 36th Conference on.* IEEE, 2011, pp. 101–108.

[47] P. Papadimitratos, A. De La Fortelle, K. Evenssen, R. Brignolo, and S. Cosenza, “Vehicular communication systems: Enabling technologies, applications, and future outlook on intelligent transportation,” *IEEE Communications Magazine*, vol. 47, no. 11, 2009.

[48] E. A. Lee, “Cyber-physical systems-are computing foundations adequate,” in *Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap*, vol. 2, 2006.

[49] ——, “Cyber physical systems: Design challenges,” in *Object oriented real-time distributed computing (isorc), 2008 11th ieee international symposium on.* IEEE, 2008, pp. 363–369.

[50] W. Wolf, “Cyber-physical systems,” *Computer*, vol. 42, no. 3, pp. 88–89, 2009.

[51] S. Chakraborty, M. A. Al Faruque, W. Chang, D. Goswami, M. Wolf, and Q. Zhu, “Automotive cyber-physical systems: A tutorial introduction,” *IEEE Design & Test*, vol. 33, no. 4, pp. 92–108, 2016.

[52] W. Chang and S. Chakraborty, “Resource-aware automotive control systems design: A cyber-physical systems approach,” *Foundations and Trends in Electronic Design Automation*, vol. 10, no. 4, pp. 249–369, 2016. [Online]. Available: https://doi.org/10.1561/1000000045

[53] D. Roy, L. Zhang, W. Chang, and S. Chakraborty, "Automated synthesis of cyber-physical systems from joint controller/architecture specifications," in *Specification and Design Languages (FDL), 2016 Forum on.* IEEE, 2016, pp. 1–8.

[54] D. Roy, L. Zhang, W. Chang, D. Goswami, and S. Chakraborty, "Multi-objective co-optimization of flexray-based distributed control systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2016 IEEE.* IEEE, 2016, pp. 1–12.

[55] D. Goswami, M. Lukasiewycz, R. Schneider, and S. Chakraborty, "Time-triggered implementations of mixed-criticality automotive software," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012.* IEEE, 2012, pp. 1227–1232.

[56] D. Goswami, R. Schneider, and S. Chakraborty, "Re-engineering cyber-physical control applications for hybrid communication protocols," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011.* IEEE, 2011, pp. 1–6.

[57] M. Lukasiewycz, R. Schneider, D. Goswami, and S. Chakraborty, "Modular scheduling of distributed heterogeneous time-triggered automotive systems," in *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific.* IEEE, 2012, pp. 665–670.

[58] E. Bini and A. Cervin, "Delay-aware period assignment in control systems," in *Real-Time Systems Symposium, 2008.* IEEE, 2008, pp. 291–300.

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

[60] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei, "Timing analysis of the flexray communication protocol," in *Real-Time Systems, 2006. 18th Euromicro Conference on.* IEEE, 2006, pp. 11–pp.

[61] U. D. Bordoloi, B. Tanasa, P. Eles, and Z. Peng, "On the timing analysis of the dynamic segment of flexray," in *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on.* IEEE, 2012, pp. 94–101.

[62] J. Diemer, D. Thiele, and R. Ernst, "Formal worst-case timing analysis of ethernet topologies with strict-priority and avb switching," in *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on.* IEEE, 2012, pp. 1–10.

[63] R. Schneider, L. Zhang, D. Goswami, A. Masrur, and S. Chakraborty, "Compositional analysis of switched ethernet topologies," in *Proceedings of the Conference on Design, Automation and Test in Europe.* EDA Consortium, 2013, pp. 1099–1104.

[64] F. Reimann, S. Graf, F. Streit, M. Glaß, and J. Teich, “Timing analysis of ethernet avb-based automotive e/e architectures,” in *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on.* IEEE, 2013, pp. 1–8.

[65] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse, “System architecture evaluation using modular performance analysis: a case study,” *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 8, no. 6, pp. 649–667, 2006.

[66] A. Hagiescu, U. D. Bordoloi, S. Chakraborty, P. Sampath, P. V. V. Ganesan, and S. Ramesh, “Performance analysis of flexray-based ecu networks,” in *Design Automation Conference, 2007. DAC’07. 44th ACM/IEEE.* IEEE, 2007, pp. 284–289.

[67] D. B. Chokshi and P. Bhaduri, “Performance analysis of flexray-based systems using real-time calculus, revisited,” in *Proceedings of the 2010 ACM Symposium on Applied Computing.* ACM, 2010, pp. 351–356.

[68] M. Lukasiewycz, M. Glaß, J. Teich, and P. Milbredt, “Flexray schedule optimization of the static segment,” in *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis.* ACM, 2009, pp. 363–372.

[69] H. Zeng, M. Di Natale, A. Ghosal, and A. Sangiovanni-Vincentelli, “Schedule optimization of time-triggered systems communicating over the flexray static segment,” *IEEE Transactions on Industrial Informatics*, vol. 7, no. 1, pp. 1–17, 2011.

[70] R. Schneider, D. Goswami, S. Chakraborty, U. Bordoloi, P. Eles, and Z. Peng, “Quantifying notions of extensibility in flexray schedule synthesis,” *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 19, no. 4, p. 32, 2014.

[71] P. Pop, P. Eles, Z. Peng, and T. Pop, “Scheduling and mapping in an incremental design methodology for distributed real-time embedded systems,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 12, no. 8, pp. 793–811, 2004.

[72] W. Zheng, J. Chong, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli, “Extensible and scalable time triggered scheduling,” in *Application of Concurrency to System Design, 2005. ACSD 2005. Fifth International Conference on.* IEEE, 2005, pp. 132–141.

[73] W. Chang, D. Roy, L. Zhang, and S. Chakraborty, “Model-based design of resource-efficient automotive control software,” in *Proceedings of the 35th International Conference on Computer-Aided Design.* ACM, 2016, p. 34.

[74] W. Chang, D. Goswami, S. Chakraborty, L. Ju, C. J. Xue, and S. Andalam, “Memory-aware embedded control systems design,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 4, pp. 586–599, 2017.

[75] P. Mundhenk, G. Tibba, L. Zhang, F. Reimann, D. Roy, and S. Chakraborty, “Dynamic platforms for uncertainty management in future automotive e/e architectures,” in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 2017, p. 15.

[76] D. Gangadharan, J. H. Kim, O. Sokolsky, B. Kim, C.-W. Lin, S. Shiraishi, and I. Lee, “Platform-based plug and play of automotive safety features: Challenges and directions,” in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2016 IEEE 22nd International Conference on*. IEEE, 2016, pp. 76–84.

[77] J. Teich, “Hardware/software codesign: The past, the present, and predicting the future,” *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1411–1430, 2012.

[78] S. Samii, A. Cervin, P. Eles, and Z. Peng, “Integrated scheduling and synthesis of control applications on distributed embedded systems,” in *Proceedings of the conference on design, automation and test in Europe*. European Design and Automation Association, 2009, pp. 57–62.

[79] A. Aminifar, P. Eles, Z. Peng, and A. Cervin, “Control-quality driven design of cyber-physical systems with robustness guarantees,” in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 1093–1098.

[80] M. Buechel, J. Frtunikj, K. Becker, S. Sommer, C. Buckl, M. Armbruster, A. Marek, A. Zirkler, C. Klein, and A. Knoll, “An automated electric vehicle prototype showing new trends in automotive architectures,” in *Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on*. IEEE, 2015, pp. 1274–1279.

[81] F. Sagstetter, M. Lukasiewycz, S. Steinhorst, M. Wolf, A. Bouard, W. R. Harris, S. Jha, T. Peyrin, A. Poschmann, and S. Chakraborty, “Security challenges in automotive hardware/software architecture design,” in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 458–463.

[82] P. Mundhenk, S. Steinhorst, M. Lukasiewycz, S. A. Fahmy, and S. Chakraborty, “Lightweight authentication for secure automotive networks,” in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE ’15. San Jose, CA, USA: EDA Consortium, 2015, pp. 285–288. [Online]. Available: http://dl.acm.org/citation.cfm?id=2755753.2755816

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

[84] H. Kopetz and G. Grunsteidl, “Ttp-a time-triggered protocol for fault-tolerant real-time systems,” in *Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on*. IEEE, 1993, pp. 524–533.

[85] F. Sagstetter, M. Lukasiewycz, and S. Chakraborty, “Schedule integration for time-triggered systems,” in *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific.* IEEE, 2013, pp. 53–58.

[86] P. Tabuada, “Event-triggered real-time scheduling of stabilizing control tasks,” *IEEE Transactions on Automatic Control*, vol. 52, no. 9, pp. 1680–1685, 2007.

[87] D. Goswami, R. Schneider, and S. Chakraborty, “Co-design of cyber-physical systems via controllers with flexible delay constraints,” in *Proceedings of the 16th Asia and South Pacific Design Automation Conference.* IEEE Press, 2011, pp. 225–230.

[88] “Gurobi optimization,” http://www.gurobi.com, accessed: 2017-06-28.

[89] W. Steiner, “Synthesis of static communication schedules for mixed-criticality systems,” in *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2011 14th IEEE International Symposium on.* IEEE, 2011, pp. 11–18.

[90] H. Kopetz and G. Bauer, “The time-triggered architecture,” *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[91] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, “The time-triggered ethernet (tte) design,” in *Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005. Eighth IEEE International Symposium on.* IEEE, 2005, pp. 22–33.

[92] R. Schneider, D. Goswami, S. Zafar, M. Lukasiewycz, and S. Chakraborty, “Constraint-driven synthesis and tool-support for flexray-based automotive control systems,” in *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis.* ACM, 2011, pp. 139–148.

[93] Z. Hanzalek, P. Burget, and P. Sucha, “Profinet io irt message scheduling with temporal constraints,” *IEEE Transactions on Industrial Informatics*, vol. 6, no. 3, pp. 369–380, 2010.

[94] D. Tamas-Selicean, P. Pop, and W. Steiner, “Synthesis of communication schedules for ttethernet-based mixed-criticality systems,” in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis.* ACM, 2012, pp. 473–482.

[95] H. P. Williams, *Model building in mathematical programming.* John Wiley & Sons, 2013.

[96] J.-W. Yoo, Y. Lee, D. Kim, and K. Park, “An android-based automotive middleware architecture for plug-and-play of applications,” in *Open Systems (ICOS), 2012 IEEE Conference on.* IEEE, 2012, pp. 1–6.

[97] C. Buckl, M. Geisinger, D. Gulati, F. J. Ruiz-Bertol, and A. Knoll, “Chromosome: A run-time environment for plug & play-capable embedded real-time systems,” *ACM SIGBED Review*, vol. 11, no. 3, pp. 36–39, 2014.

[98] A. Zeeb, “Plug and play solution for autosar software components,” *ATZelektronik worldwide*, vol. 7, no. 1, pp. 16–21, 2012.

[99] L. Zhang, D. Goswami, R. Schneider, and S. Chakraborty, “Task-and network-level schedule co-synthesis of ethernet-based time-triggered systems,” in *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific.* IEEE, 2014, pp. 119–124.

[100] S. S. Craciunas and R. S. Oliver, “Smt-based task-and network-level static schedule generation for time-triggered networked systems,” in *Proceedings of the 22nd International Conference on Real-Time Networks and Systems.* ACM, 2014, p. 45.

[101] M. Gutiérrez, W. Steiner, R. Dobrin, and S. Punnekkat, “A configuration agent based on the time-triggered paradigm for real-time networks,” in *Factory Communication Systems (WFCS), 2015 IEEE World Conference on.* IEEE, 2015, pp. 1–4.

[102] ——, “Learning the parameters of periodic traffic based on network measurements,” in *Measurements & Networking (M&N), 2015 IEEE International Workshop on.* IEEE, 2015, pp. 1–6.

[103] H. Martorell, J.-C. Fabre, M. Roy, and R. Valentin, “Towards dynamic updates in autosar,” in *SAFECOMP 2013-Workshop CARS (2nd Workshop on Critical Automotive applications: Robustness & Safety) of the 32nd International Conference on Computer Safety, Reliability and Security*, 2013, p. NA.

[104] J. Frtunikj, V. Rupanov, A. Camek, C. Buckl, and A. Knoll, “A safety aware run-time environment for adaptive automotive control systems,” *Embedded real-time software and systems (ERTS2)*, vol. 3, 2014.

[105] “Gecode generic constraint development environment,” http://www.gecode.org, accessed: 2017-06-28.

[106] “Github ole christian eidheim,” https://github.com/eidheim, accessed: 2017-06-28.

[107] W. Chang and S. Chakraborty, “Resource-aware automotive control systems design: A cyber-physical systems approach,” *Foundations and Trends in Electronic Design Automation*, vol. 10, no. 4, pp. 249–369, 2016. [Online]. Available: https://doi.org/10.1561/1000000045

[108] A. M. Annaswamy, D. Soudbakhsh, R. Schneider, D. Goswami, and S. Chakraborty, “Arbitrated network control systems: A co-design of control and platform for cyber-physical systems,” in *Control of cyber-physical systems.* Springer International Publishing, 2013, pp. 339–356.

[109] D. Majumdar, L. Zhang, P. Bhaduri, and S. Chakraborty, "Reconfigurable communication middleware for flex ray-based distributed embedded systems," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2015 IEEE 21st International Conference on.* IEEE, 2015, pp. 159–166.

[110] E. Bini and A. Cervin, "Delay-aware period assignment in control systems," in *IEEE RTSS*, 2008.

[111] A. Cervin and P. Alriksson, "Optimal on-line scheduling of multiple control tasks: A case study," in *ECRTS*, 2006.

[112] R. Castane, P. Martí, M. Velasco, and A. Cervin, "Resource management for control tasks based on the transient dynamics of closed-loop systems," in *ECRTS*, 2006.

[113] R. Postoyan, P. Tabuada, D. Nesic, and A. A. Martinez, "Event-triggered and self-triggered stabilization of distributed networked control systems," in *IEEE Control Decision Conference (CDC)*, 2011.

[114] M. M. Jr. and P. Tabuada, "On event-triggered and self-triggered control over sensor/actuator networks," in *IEEE Control Decision Conference (CDC)*, 2008.

[115] F. Zhang, K. Szwaykowska, W. Wolf, and V. J. Mooney, "Task scheduling for control oriented requirements for Cyber-Physical Systems," in *IEEE Real-Time Systems Symposium (RTSS)*, 2008.

[116] P. Naghshtabrizi and J. Hespanha, "Analysis of distributed control systems with shared communication and computation resource," in *ACC*, 2009.

[117] R. Alur, A. D'Innocenzo, K. H. Johansson, G. J. Pappas, and G. Weiss, "Compositional modeling and analysis of multi-hop control networks," *IEEE Transactions on Automatic Control*, vol. 56, no. 10, pp. 2345–2357, 2011.

[118] M. Pajic, S. Sundaram, G. J. Pappas, and R. Mangharam, "The wireless control network: A new approach for control over networks," *IEEE Transactions on Automatic Control*, vol. 56, no. 10, pp. 2305–2318, 2011.

[119] X. Wang and M. Lemmon, "Event-triggering in distributed networked control systems," *IEEE Transactions on Automatic Control*, vol. 56, no. 3, pp. 586–601, 2011.

[120] S. Samii, P. Eles, Z. Peng, and A. Cervin, "Design optimization and synthesis of FlexRay parameters for embedded control applications," in *DELTA*, 2011.

[121] A. Masrur, D. Goswami, S. Chakraborty, J.-J. Chen, A. Annaswamy, and A. Banerjee, "Timing analysis of cyber-physical applications for hybrid communication protocols," in *DATE*, 2012.

[122] D. Goswami, M. Lukasiewycz, R. Schneider, and S. Chakraborty, "Time-triggered implementations of mixed-criticality automotive software," in *DATE*, 2012.

[123] A. Masrur, D. Goswami, R. Schneider, H. Voit, A. Annaswamy, and S. Chakraborty, "Schedulability analysis of distributed cyber-physical applications on mixed time-/event-triggered bus architectures with retransmissions," in *DATE*, 2012.

[124] D. Goswami, R. Schneider, and S. Chakraborty, "Co-design of Cyber-Physical Systems via controllers with flexible delay constraints," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2011.

[125] R. Schneider, D. Goswami, S. Zafar, M. Lukasiewycz, and S. Chakraborty, "Constraint-driven synthesis and tool-support for FlexRay-Based automotive control systems," in *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2011.

[126] D. Goswami, R. Schneider, and S. Chakraborty, "Re-engineering cyber-physical control applications for hybrid communication protocols," in *DATE*, 2011.

[127] ——, "Relaxing signal delay constraints in distributed embedded controllers," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 6, pp. 2337–2345, 2014.

[128] Z. Sun and S. S. Ge, *Stability theory of switched dynamical systems.* Springer Science & Business Media, 2011.

[129] O. Mason and R. Shorten, "On common quadratic lyapunov functions for stable discrete-time lti systems," *IMA Journal of Applied Mathematics*, vol. 69, no. 3, pp. 271–283, 2004.

List of Figures

List of Tables

List of Acronyms

ABS	Anti-lock Braking System.
ACC	Adaptive Cruise Control.
ADAS	Advanced Driver Assistance System.
AFDX	Avionics Full Duplex Switched Ethernet.
AUTOSAR	Automotive Open System Architecture.
AVB	Audio Video Bridging.
BE	Best-Effort.
CAN	Controller Area Network.
CAN FD	CAN with Flexible Date-Rate.
COTS	Commercial-Off-The-Shelf.
CPS	Cyber-Physical System.
CQLF	Common Quadratic Lyapunov Function.
CSMA/CD	Carrier Sense Multiple Access/Collision Detection.
CSMA/CR	Carrier Sense Multiple Access/Collision Resolution.
DDS	Data Distribution Service.
E/E	Electrical/Electronic.
ECU	Electronic Control Unit.
ESP	Electronic Stability Program.
ET	Event-Triggered.
FTDMA	Flexible Time Division Multiple Access.

GPS	Global Positioning System.
HMI	Human Machine Interface.
HTTP	Hypertext Transfer Protocol.
I/O	Input/Output.
IP	Internet Protocol.
IRT	Isochronous Real-Time.
ITS	Intelligent Transportation System.
LDW	Lane Departure Warning.
LIDAR	Light Detection and Ranging.
LIN	Local Interconnect Network.
LMI	Linear Matrix Inequity.
LTI	Linear Time-Invariant.
MAC	Media Access Control.
MILP	Mixed Integer Linear Program.
MIP	Mixed Integer Program.
MOST	Media Oriented Systems Transport.
NCS	Networked Control System.
NRT	Non-Real-Time.
OEM	Original Equipment Manufacturer.
OS	Operating System.
OSEK	Offene Systeme und deren Schnittstellen fur die Elektronik in Kraftfahrzeugen.
PDU	Protocol Data Unit.
PHY	Physical.
PnP	Plug-and-Play.
PSO	Particle Swarm Optimization.
RC	Rate-Constrained.
RT	Real-Time.
RTE	Runtime Environment.
SIL	Safety Integration Level.
SMT	Satisfiability Modulo Theories.
SOA	Service Oriented Architecture.
SOC	Service Oriented Communications.

SOME/IP	Scalable service-oriented Middleware over IP.
SSL/TLS	Secure Sockets Layer/Transport Layer Security.
TCP	Transmission Control Protocol.
TCS	Traction Control System.
TDMA	Time Division Multiple Access.
TSN	Time-Sensitive Networking.
TT	Time-Triggered.
TTCAN	Time Triggered CAN.
TTP	Time-Triggered Protocol.
UDP	User Datagram Protocol.
VDX	Vehicle Distribution eXecutive.
VFB	Virtual Functional Bus.
WCET	Worst-Case Execution Time.