Scheduling Event-Triggered and Time-Triggered Applications with Optimal Reliability and Predictability on Networked Multi-Core Chips

DISSERTATION
zur Erlangung des akademischen Grades
eines Doktors der Ingenieurwissenschaften (Dr.-Ing.)

vorgelegt Dissertation von:

Ayman Murshed

eingereicht bei der Naturwissenschaftlich-Technischen Fakultät
der Universität Siegen
Siegen– Januar 2018

Gedruckt auf alterungsbeständigem holz- und säurefreiem Papier.

Betreuer und erster Gutachter
Prof. Dr. Roman Obermaisser, Universität Siegen

Zweiter Gutachter
Prof. Dr. Kristof Van Laerhoven, Universität Siegen

Prüfungskommission:
Prof. Dr. Roman Obermaisser
Prof. Dr. Kristof Van Laerhoven
Prof. Dr. Roland Wismüller
Prof. Dr. Marcin Grzegorzek (Vorsitz der Prüfungskommission)

Tag der mündlichen Prüfung: 23. March 2018

Printed with the support of the German Academic Exchange Service.

In loving memory of my father

To my mother

To my beloved wife and our precious kids Ghazel and Nageeb

Acknowledgements

I would like to express my sincere gratitude to my advisor Prof. Dr-Ing. Roman Obermaisser for the continuous support of my Ph.D. research, for his patience, motivation, and constructive criticism throughout my study. His guidance helped me in all the time of research and writing of this thesis.

I would especially like to thank my family. My sincere greeting is to my late father, who always believed in my ability to be successful in the academic arena. You are gone but your belief in me has made this journey possible. I would also like to thank my mother and my brother Amjed for their support and good wishes. My wife, Sajaa has been extremely supportive of me throughout this entire process and has made countless sacrifices to help me get to this point. My children, Ghazel and Nageeb, have continually provided the requisite breaks from philosophy and the motivation to finish my degree.

I would gratefully acknowledge the funding received towards my PhD from the German Academic Exchange Service (DAAD).

Last but not the least, my gratitude is extended to all my friends and colleagues thank you for your motivation and encouragement.

Abstract

Multi-core processors are gaining increasing importance in safety-relevant embedded real-time systems, where temporal guarantees must be ensured despite the sharing of on-chip resources such as processor cores and networks-on-chip (NoC). At the same time, many applications comprise workloads with different timing models including time-triggered and event-triggered communication.

The first contribution is a scheduling model based on Mixed Integer Linear Programming (MILP) supporting the allocation of computational jobs to processing cores as well as the scheduling of messages and the selection of paths on NoC. The model supports dependencies between computational jobs and it combines both time-triggered and event-triggered messages. Phase alignment of time-triggered messages is performed while avoiding collisions between time-triggered messages and satisfying bandwidth constraints for event-triggered messages. Example scenarios are solved optimally using the IBM CPLEX optimizer yielding minimal computational and communication latencies.

Real-time communication and reliability are two important requirements in the development of safety-critical embedded systems, which benefit from the inherent fault isolation and temporal predictability of time-triggered networks. These systems depend on redundant communication schedules that contain global time-based information of message transmissions with conflict-free paths through the switches. In these systems, the use of redundancy to handle communication errors requires the preallocation of communication resources. The second contribution introduces a novel scheduler for redundant time-triggered networks that assigns messages to redundant paths. The scheduler considers the link reliability along with physical and logical models and produces a schedule where each message is assigned to two different paths along the switches. We also discuss and validate the approach with results from a prototype implementation.

SoS consist of complex interconnections of large numbers of networked embedded systems that are characterized by operational and managerial independence of constituent systems, geographical separation, and emergent behavior in a constantly changing environment. The support for real-time communication is crucial for many SoS application areas such as medical, business, and military systems. The third contribution is a conceptual model and a scheduling algorithm for supporting real-time requirements in SoS. The search for a feasible schedule is computed incrementally upon the introduction of new applications in the SoS. The distributed computation of the schedule using the different constituent systems considers the lack of global knowledge and control in the SoS, while also reducing the overall scheduling time. Concurrent scheduling activities are supported to deal with the uncoordinated and possibly simultaneous introduction of multiple applications.

The dissertation introduces also a simulation framework with real-time support of SoS that supports high-level scheduling as well as low-level scheduling for each constituent system. A time-triggered Ethernet (TTEthernet) simulation framework was extended by adding a scheduler layer to perform incremental scheduling among Constituent System Managers (CSMs). The simulation framework enabled the evaluation of the proposed algorithms in terms of schedulability, run-time, and worst-case latency for time-triggered and rate-constrained messages.

Kurzfassung

Mehrkernprozessoren gewinnen zunehmend an Bedeutung in sicherheitsrelevanten eingebetteten Echtzeitsystemen, bei denen trotz der gemeinsamen Nutzung von On-Chip-Ressourcen wie Prozessorkernen und On-Chip-Netzwerken zeitliche Garantien gewährleistet sein müssen. Gleichzeitig umfassen viele Anwendungen Arbeitsbelastungen mit unterschiedlichen Timing-Modellen, einschließlich zeitgesteuerter und ereignisgesteuerter Kommunikation.

Der erste Beitrag der Dissertation ist ein Planungsmodell, das auf der gemischt-ganzzahligen linearen Programmierung basiert und die Zuweisung von Rechenaufträgen an Prozessorkerne sowie die Planung von Nachrichten und die Auswahl von Wegen auf NoCs unterstützt. Das Modell unterstützt Abhängigkeiten zwischen Rechenjobs und kombiniert sowohl zeitgesteuerte als auch ereignisgesteuerte Nachrichten. Die Phasenausrichtung zeitgesteuerter Nachrichten wird durchgeführt, während Kollisionen zwischen zeitgesteuerten Nachrichten und die Verletzung von Bandbreitenbeschränkungen für ereignisgesteuerte Nachrichten vermieden werden. Beispielszenarien werden optimal mit dem IBM CPLEX-Optimierer gelöst, wobei minimale Rechen- und Kommunikationslatenzen garantiert werden.

Echtzeitkommunikation und Zuverlässigkeit sind zwei wichtige Anforderungen bei der Entwicklung sicherheitskritischer eingebetteter Systeme, die von der inhärenten Fehlerisolierung und zeitlichen Vorhersagbarkeit zeitgesteuerter Netzwerke profitieren. Als Grundlage für Fehlertoleranz benötigen diese Systeme außerdem redundante Kommunikationspläne, die globale zeitbasierte Informationen von mehrfachen Nachrichtenübertragungen mit konfliktfreien Pfaden durch die Switches enthalten. In diesen Systemen erfordert die Verwendung von Redundanz zur Behandlung von Kommunikationsfehlern die Vorbelegung von Kommunikationsressourcen. Der zweite Beitrag der Dissertation stellt einen neuartigen Scheduler für redundante zeitgesteuerte Netzwerke vor, der Nachrichten redundanten Pfaden zuweist. Der Scheduler berücksichtigt die Verbindungszuverlässigkeit zusammen mit physischen und logischen Modellen und erstellt einen Zeitplan, bei dem jede Nachricht zwei verschiedenen Pfaden entlang der Switches zugewiesen wird. Wir diskutieren und validieren den Ansatz mit den Ergebnissen einer Prototypimplementierung.

Systeme von Systemen (SoS) bestehen aus komplexen Zusammenschaltungen einer großen Anzahl von vernetzten eingebetteten Systemen, die durch betriebliche Unabhängigkeit von Teilsystemen, geografische Trennung und emergentes Verhalten in einer sich ständig verändernden Umgebung gekennzeichnet sind. Die Unterstützung für Echtzeitkommunikation ist für viele Anwendungsbereiche wie medizinische, geschäftliche und militärische Systeme von entscheidender Bedeutung. Der dritte Beitrag der Dissertation ist ein konzeptionelles Modell und ein Planungsalgorithmus zur Unterstützung von Echtzeitanforderungen in SoS. Die Suche nach einem realisierbaren Zeitplan wird schrittweise nach der Einführung neuer Anwendungen im SoS berechnet. Die verteilte Berechnung des Zeitplans unter Verwendung der verschiedenen Teilsysteme berücksichtigt den Mangel an globalem Wissen und Kontrolle im SoS, während gleichzeitig die Gesamtplanungszeit verringert wird. Gleichzeitige Terminierungsaktivitäten werden unterstützt, um die unkoordinierte und möglicherweise gleichzeitige Einführung mehrerer Anwendungen zu bewältigen.

Die Dissertation stellt auch ein Simulationsframework mit Echtzeit-Unterstützung von SoS vor, das sowohl die High-Level-Planung als auch die Low-Level-Planung für jedes Teilsystem unterstützt. Ein Simulationsframework für zeitgesteuertes Ethernet (TTEthernet) wurde um eine Scheduler-Schicht erweitert, um eine inkrementelle Planung unter Constituent System Managern (CSMs) durchzuführen. Das Simulationsframework ermöglichte die Evaluierung der vorgeschlagenen Algorithmen hinsichtlich der Planbarkeit, der Laufzeit und der Worst-Case-Latenz für zeitgesteuerte und ratenbeschränkte Nachrichten.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

The advances of the semiconductor industry resulted in a trend towards multi-core processors in safety-critical real-time systems. The performance improvement of single-core processors is roughly proportional to the square root of the increase in the number of transistors, while multi-core processors promise a linear performance gain [BC11]. The required parallelism of the application software is common in most embedded systems, where numerous concurrent activities are required such as sensing, control, actuation and diagnostic functions.

A major challenge towards the use of multi-core processors in real-time systems is the temporal interference between cores. Cores require access to shared resources such as input/output devices, external memory, and the on-chip interconnect. The dynamic resolving of this resource contention significantly complicates Worst-Case Execution Time (WCET) analysis and results in pessimistic upper bounds of execution times [ABD+13]. Experimental evaluations have shown that the WCET on a commercial off-the-shelf (COTS) multi-core processor can be multiple times higher than the WCET of the same application on a single core without other interfering cores [NP12].

At present, two different approaches are perceived to enable the use of multi-core processors in safety-critical real-time systems. Firstly, development methods for deploying COTS multi-core processors in safety-relevant systems have been introduced, e.g., by performing probabilistic WCET analysis [KQA+14] and randomization techniques [Dav13]. Secondly, several researchers propose multi-core architectures targeted towards temporal predictability, whereas COTS processors for consumer applications are optimized for average execution times. These real-time architectures are based on Time-Triggered Network-on-Chips (TTNoCs) with short and predictable WCET. Examples are the GENESYS MPSoC using the time-triggered Network-on-Chip (NoC) [OKP10], COMPSoC with the AEthereal NoC [GAC+13] and PARMERASA [UBG+13].

Likewise, many distributed embedded systems are based on multiple clusters and time-triggered networks with complex topologies. Examples are electronic systems in industrial control, the automotive domain and avionics. In these systems, a large number of end-systems is required, which are connected to switches using stars while switches themselves are interconnected using different topologies [DT03], [DYN03a]. Furthermore, there is a trend to combine complex network topologies with different timing models including rate-constrained and periodic communication. Examples are large-scale systems based on Time Sensitive Networking (TSN) [IEE15b] and Time-Triggered Ethernet (TTEthernet) [AS611].

The TTNoCs in multi-core architectures as well as in distributed embedded systems depend on scheduling and allocation algorithms in order to guarantee that the communication and computational activities meet the deadlines. In case of NoCs using Time-division Multiple Access (TDMA), conflict-free sending slots need to be assigned to the message-exchanges between the cores. For event-triggered communication activities, worst-case delays imposed by competing messages with given predefined minimum interarrival times need to be analyzed.

In general, scheduling and allocation techniques from the area of distributed systems cannot directly be applied to multi-core architectures due to differences between on-chip and off-chip networks. These differences include the differences in the network topologies (e.g., regular distributed topologies vs. customized on-chip topologies), different routing protocols (source-based vs. distributed routing) and control schemes (e.g., granularity of communication, interleaving of virtual networks).

In order to ensure reliability in time-triggered networks, there should be a mechanism to guarantee the timely message delivery in the presence of communication link faults. Generally, most of the common faults can be masked using redundancy techniques [BK00]. Depending on the system type, there are two main techniques to overcome communication errors: temporal and spatial redundancy [NSL09]. The first technique transmits messages over the same link in different time intervals, while the second method sends the message copies through different links.

Furthermore, the field of embedded systems is faced with the trend of an increasing interconnection of independently developed embedded systems to each other and to the cloud. The resulting Systems-of-Systems (SoSs) are networked together for a period of time to achieve a certain higher goal [Jam09]. Examples of SoSs include smart cities [ZBC$^+$14], intelligent factories [Jaz14] and integrated healthcare systems [WCBM07]. SoSs are characterized by operational and managerial independence of constituent systems, geographical distribution, emergent behavior and evolutionary development processes [Mai98]. In addition, many SoSs depend on support for stringent real-time requirements for time-critical application services. Examples are medical monitoring and telemedicine in healthcare systems.

Real-time support in SoSs is an open research challenge due to the lack of central control as well as the evolving and dynamic nature of the interactions between the constituent systems. In monolithic systems, the dynamic introduction of new applications is performed using schedulability tests in order to ensure that accepted applications meet their real-time requirements and new applications do not affect existing ones.

1.1 Research Scope

This thesis presents time-triggered scheduling models and optimal scheduling algorithms that support real-time and reliability requirements for NoCs as well as SoSs. This thesis addresses time-triggered systems, because the temporal interference between the cores of event-triggered multi-core processors significantly complicates the analysis of WCETs. For example, due to the interference at the level of input/output devices, external memory and on-chip interconnects, event-triggered multi-core processors are not recommended in safety-critical avionic systems [CASTCA16].

Time-triggered networks exhibit a predictable temporal behavior of communication systems as well as fault containment where end-systems are working based on a permitted behavior with respect to a global time base. The challenge is how to guarantee the timely message delivery in a multi-hop time-triggered networks with the presence of communication link faults. Moreover, some applications have event-triggered communication in addition to time-triggered, which requires minimizing the end-to-end delay of event-triggered messages while ensuring the timing requirements for time-triggered messages. Previous research work either presented temporal redundancy mechanism which requires schedule change when a message dropped or did not introduce a complete model that provides a reliable schedule; i.e., including job allocation, precedence constraints, and link failure.

The end-to-end communication with real-time requirements in SoS imposes a number of challenges such as managerial independence of constituent systems, emergent behavior, and evolutionary development processes. The scheduling of these large-scale networked embedded systems requires a decentralized algorithm in which each constituent system is only aware of its own resources at the same time the introduction of new applications requires a model that considers the allocation of reserved resources for the previous applications in order to avoid resource contention with the new applications to be scheduled.

1.2 Thesis Contributions

The scientific contributions of this thesis beyond the state of the art are as follows:

1. ***Scheduler for Distributed Systems with Time-Triggered Networks:*** We introduce a system model for distributed systems deployed with time-triggered networks that supports time-triggered and event-triggered communications. The system model is mapped to a Mixed-Integer Linear Programming (MILP) problem for optimizing the allocation and scheduling of computational and communication activities. Constraints ensure correct allocations based on a given distributed system architecture and application model (e.g., dependencies between communication and computational activities, limited connectivity between routers). The MILP problem also expresses real-time constraints and avoids collisions between time-triggered messages.

2. ***Extension of Time-Triggered Networks scheduler for reliability:*** We extend the scheduler model from (1) to optimize the reliability as part of the objective function. This is done by generating redundant messages each with different paths. As a consequence, communication reliability is improved compared to the failure rates of individual communication links.

3. ***Scheduler for SoSs with real-time requrements*:** We presents an SoS architecture with support for real-time requirements based on managed traffic and dynamic configuration. Each constituent system is equipped with a Constituent System Manager (CSM), which not only configures the local communication networks within the constituent systems but also interacts with the CSMs of other constituent systems and the backbone infrastructure of the SoS to establish resource reservations. This is done by formulating an incremental, distributed and concurrent scheduling problem for the CSMs. The computed schedules lead to resource reservations for time-triggered communication and computational activities.

1.3 Thesis Structure

The remainder of the dissertation is structured as follows.

Chapter 2 contains the main concepts and terms that are used throughout the dissertation. First, it outlines the background of real-time system focusing on distributed systems explaining their timing models and their main classifications. Then, an introduction on real-time scheduling problem is introduced with the focus on linear programming method to solve such problems. After that, the concept of dependability and its main classifications is explained followed by an introduction on the fault hypothesis concept. Finally, an overview of SoS with its main characteristics is presented.

Chapter 3 starts with the classifications of scheduling algorithms and followed by an overview of the state of the art in the scheduling of distributed systems. Afterwards, the research gaps in the state of the art are outlined with the proposed solutions.

In Chapter 4 the system model for safety critical embedded systems based on multi-core architecture is introduced. The introduced model is then mapped into a scheduling model where jobs are mapped to end-systems and messages of different criticality are optimally scheduled. Finally, example scenarios and optimization results are presented.

Chapter 5 provides an optimized model presented in the previous chapter where the search space of the model is reduced and consequently scheduling execution time is less compared to the previous model. After that, the reliability concept in time-triggered embedded systems is highlighted focusing on the parallel-series model as a redundancy technique. The explained model is then mapped into a scheduling model that integrates a redundant model for safety critical embedded system based on multi-core architecture. The proposed model is then evaluated using example scenarios.

Chapter 6 presents a conceptual model of an SoS. Based on the described model, a scheduling model for an SoS architecture is explained based on incremental, distributed, and concurrent scheduling principles. The proposed model is then formulated and evaluated using example scenarios.

Chapter 7 describes a simulation framework for SoS model to validate the previously described SoS scheduling model. It starts with a detailed description of the building blocks in an SoS simulation framework. Then, a simulation tool chain is presented which consists of the main processes of the simulation framework. After that, example scenarios and simulation results are presented using OPNET simulation. Finally, the previously described SoS building blocks are extended where the SoS scheduling model is integrated inside the end-system process model of the simulation.

Chapter 8 concludes the dissertation and summarizes the main outcomes of the contributions presented in the previous chapters.

Chapter 2

Concepts and Terms

This chapter provides definitions and background highlights about the main concepts and terms that are used throughout this dissertation. It starts with the fundamental concept of real-time system including its important characteristics and its main components. Then, an overview of the basic principles of a distributed real-time system is presented explaining its timing models in addition to its significant classifications. After that, a general review about the timing concept in real-time system followed by a brief notion about real-time scheduling and linear programming. Next, the attributes and means of the dependability concept are outlined with a background about the fault-hypothesis. Finally, definitions and a quick review of the term SoS are given highlighting its main characteristics.

2.1 Real-time Systems

A system is a collection of dependent components that interact to produce a desired emergent behavior. The functionalities of a system depend on the structural parts of their components, connections and relationships between the interacting components, and the behaviors of the overall system.

An embedded system is a combination of hardware and software which is designed to accomplish a particular task or several tasks. There are many definitions for embedded systems. [Kam11] defined an embedded system as a system that has two main components; particularly computer-hardware and embedded software that are combined together for a specific application or part of a larger system. In another definition by [Shi09], an embedded system is a combination of special-purpose hardware and embedded software, that may consist of electro-mechanical parts, which is designed to perform a specific function.

Real-time systems have a number of characteristics that are different compared to other computer systems. The correctness of real-time systems does not only depend on the delivery of the logical results of computation but also on the time at which these results are delivered [Sta88], [Kop11]. The important characteristic of these systems is the real-time operation which is defined by the German industry standard DIN 44300 [Kav92] as ‘the operating mode of computer systems in which the programs for the processing of data arriving from the outside are permanently ready, so that their results will be available within predetermined periods of time; the arrival times of the data can be randomly distributed or be already a priori determined depending on the different applications’.

A real-time application is composed of a set of tasks that have different levels of criticality. The consequence of missing deadlines in hard real-time tasks is catastrophic and may results to deadly results. On the other hand, soft real-time tasks can miss deadlines without severe damage and the system still work correctly [MJ95].

A system that consists of set of job tasks, $J = \{j_1, j_2, \ldots, j_n\}$ where their finish times are $F = \{f_1, f_2, \ldots, f_n\}$ consecutively is said to be real-time iff there exists at least one job j_i relates to J in which its execution should not be more than the given deadline d_i; i.e., $f_i < d_i$

To better understand the concept of real-time system, consider the example of a nuclear power plant monitoring system, where the nuclear core temperature and pressure are continuously monitored by the sensors of a real-time control system. Based on periodic readings from the sensors, a coolant is fed to keep the nuclear core below a certain temperature while controlling the pressure inside the reactor using a pilot-operated relief valve (PORV).

As illustrated in the above example, a real-time system changes as a function of physical time. Generally, a real-time system can be decomposed into a set of self-contained clusters, namely the computational cluster and its environment, such as controlled cluster and operator cluster as shown in Figure 2.1. In the nuclear plant monitoring system example, the temperature and pressure are part of the controlled cluster, the system that monitors the readings of pressure and temperature and invokes real-time actions is the computational cluster, and the operator cluster is represented by the observer who keeps track of the readings and actions taken by the system. Thus, a real-time computational cluster must analyze the input taken from the controlled cluster and react within a specified time interval dictated by its environment.

2.2 Distributed Real-time Systems

A distributed system is defined by [CDK05] as a collection of networked computers, in which each computer consists of hardware and software components, that communicate and coordinate together by using message transmissions. Another definition of a distributed

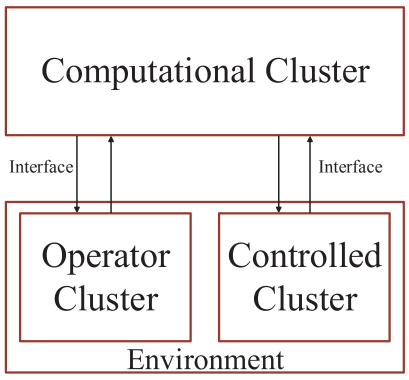


Fig. 2.1 Real-time system

system by [Tan95], is a collection of autonomous computers that collaborate together to give the perception of a single system.

A distributed real-time system performs activities that have specific time bounds. Hence, the difference between distributed real-time systems and ordinary distributed systems is the time bounds of the performed activities.

Distributed real-time systems interact with the environment through application tasks that perform activities. This is realized by reading the received data form sensors and consequently reacting by means of actuators. In order to achieve the time bounds of the aforementioned activities, both processing and communication events associated with those activities should be constrained in time domain by means of task and message scheduling 2.5.

2.2.1 Timing Models in Distributed Real-Time Systems

Due to many different and partially contradicting requirements in embedded systems, multiple timing models exist for building a communication system. Well-known trade-offs are predictability versus flexibility or resource adequacy versus best-effort strategies. According to these requirements, we distinguish between two types of communication networks [Edi12]:

- **Time-triggered communication of messages** ensures predictability and resource adequacy by a priori scheduled transmission times of periodic messages. This transmission is characterized by a *period* and a *phase* with respect to a *global time base*. By dedicating *a priori* defined bandwidth to time-triggered messages, timely delivery of all messages is guaranteed. Time-triggered NoCs (e.g., TTNoC [Pau08]) are beneficial in safety-critical systems, because they help in managing the complexity of fault-tolerance and analytical dependability models. The static schedule of a time-triggered system maximizes predictability, while the schedule in an event-triggered network unfolds dynamically at run-time depending on the occurrence of events.

- **Event-triggered communication** supports the transmission of messages that are triggered by the occurrence of significant events in the environment or inside a core. For example, a core requests the transmission of a message whenever an interrupt arrives from a sensor. We can distinguish between two types of event-triggered messages: *rate-constrained* and *best-effort* messages. Rate-constrained communication is used by applications with less stringent timing requirements, compared to time-triggered. These messages have sufficient bandwidth allocation for each transmission with defined limits for delays and temporal deviations. The arrival times of rate-constraint messages are not specifically known but they have minimum time intervals between consecutive instances and it is called the Bandwidth Allocation Gap (BAG). The best-effort transmission of messages there is no guarantee whether or when the message will arrive at the destination where they are transmitted during the idle periods where no time-triggered nor rate-constrained messages are being transmitted. Priorities determine how contention with other event-triggered messages is resolved.

Time-triggered and rate-constrained messages have unique identifiers called Virtual Links (VLs) which are virtual transmission channels that make them known while they are transmitted through the network. The communication in TTEthernet is based on VLs where each defines the necessary reserved resources of one message for end-to-end transmission [SAE]. In other words, VLs define the bandwidth allocation of the communication network in order to provide certain level of temporal message delivery.

Event-triggered networks provide flexibility and high resource utilization and suit non-critical applications very well. However, there are some NoCs such as AEthereal [GDR05] that support both event-triggered and time-triggered communication. AEthereal supports both guaranteed services based on TDMA as well as event-triggered Best Effort Services (BESs). Guaranteed services need resource reservations in order to increase resource utilization.

2.2.2 Classification of Real-time Distributed Systems

Real-time systems can be classified according to [Kop11] from different perspectives. In this thesis, three main important classifications are defined in this subsection. Based on the laxities of tasks and severity of consequences of missed deadlines, real-time systems can be classified into *hard real-time* and *soft real-time* systems. Laxity is the amount of time tolerated by a task while still meeting its deadline and avoiding severe consequences. Hard-real-time systems can also be classified according to how they react to failures. These classifications are *fail-safe* and *fail-operational* systems. In addition, real-time systems can be classified based on the triggering mechanism for activities, which can be *time-triggered* or *event-triggered*.

2.2.2.1 Hard Real-time versus Soft Real-time Systems

Hard real-time systems have strict timing constraints and a temporal failure of a critical service can lead to deadly results, human life loss, high economic loss, or extensive environmental damage. These systems have little laxity and provide full deadline compliance. Systems that incorporate this feature are called safety-critical systems. These systems require predictable, reliable, and real-time communication between the end-systems to ensure safety and reliability. These systems are found in many domains such as military applications, automotive applications, and flight control applications.

Soft real-time systems have less stringent timing constraints in which temporal failures do not lead to catastrophic results, but result in performance degradation. They have greater laxity and can tolerate certain amounts of deadline misses. In other words, the usefulness of the result is degraded as a consequence of deadline misses. Examples of these systems are multimedia applications.

2.2.2.2 Fail-safe versus Fail-operational Systems

A fail-safe system has two important characteristics: high probability of error-detection and a pre-determined safe state. When the system detects a component failure, it forces the system to a safe state to prevent damage. For example, some machines are designed to switch off operation if they detect a component failure.

A system that must continue to provide an acceptable level of services even in the occurrence of failures in order to avoid a catastrophe is called a fail-operational system. These systems require fault-tolerance mechanisms to mask component failures. For instance, computer-based flight control systems are designed with redundancy so that if one fails the other system continues operation to avoid an airplane crash.

2.2.2.3 Time-Triggered versus Event-Triggered Systems

Time-triggered systems depend on interrupts triggered by a periodic real-time clock for the communication and processing activities. A time-triggered schedule contains information about the real-time instances for every system activity, such as initiation of a job execution or message transmission. Distributed real-time systems must have a global time base where the clocks of all nodes are synchronized.

On the other hand, event-triggered systems perform communication and processing activities based on control signals created by events. These events are initiated either inside the computer system (e.g., task completion) or outside the computer system (e.g., message receipt).

2.2.3 Timing Concepts

Most hard real-time systems are based on periodic tasks which are invoked after fixed time intervals [Kop11]. The attributes of these periodic tasks must be known a priori, such as period, deadline, and required resources [HS97]. On the other hands, there are sporadic tasks where the invocation times are unknown but they have known inter-arrival times. The third type of tasks is called aperiodic and it has neither invocation times, nor inter-arrival times.

The *release time* is the instant of time at which a task becomes ready for execution [But11]

The *execution time*, also called computation time, is the time required by the processor for executing the task without interruption [But11]. The estimation of the execution time depends mainly on system resources and inter-task dependencies.

Real-time systems perform computational and communication activities with temporal constraints that must be met in order to ensure their desired behavior. A temporal constraint of a real-time activity is called a *deadline*, which denotes the latest point of time when the system should produce a correct result.

In order to guarantee the deadlines of all real-time tasks, the maximum duration latency of all computation and communication activities in all components of a real-time system must be known a priori. A WCET is defined as the maximum latency required between task initiation and task termination. Also, a Worst-Case Communication Time (WCCOM) is defined as the maximum latency required between a sending action and the receipt action [Kop11].

The *end-to-end delay* of a message in a distributed system is the duration between the time the message is injected by the sender component till the delivery at the destination component through multiple hops along routers and communication links.

2.2.4 Precedence Constraints

Certain applications consists of jobs that respect precedence relations that describes the execution order. These jobs cannot be executed in random order and have to start their execution after the completion of all its predecessor jobs. The precedence relations are described by using a Directed Acyclic Graph (DAG) where jobs are represented by nodes and precedence relations by arrows [But11]. A directed path from node J_a to node J_b, means that job J_a is a predecessor of job J_b and can be described by the notation $J_a \prec J_b$.

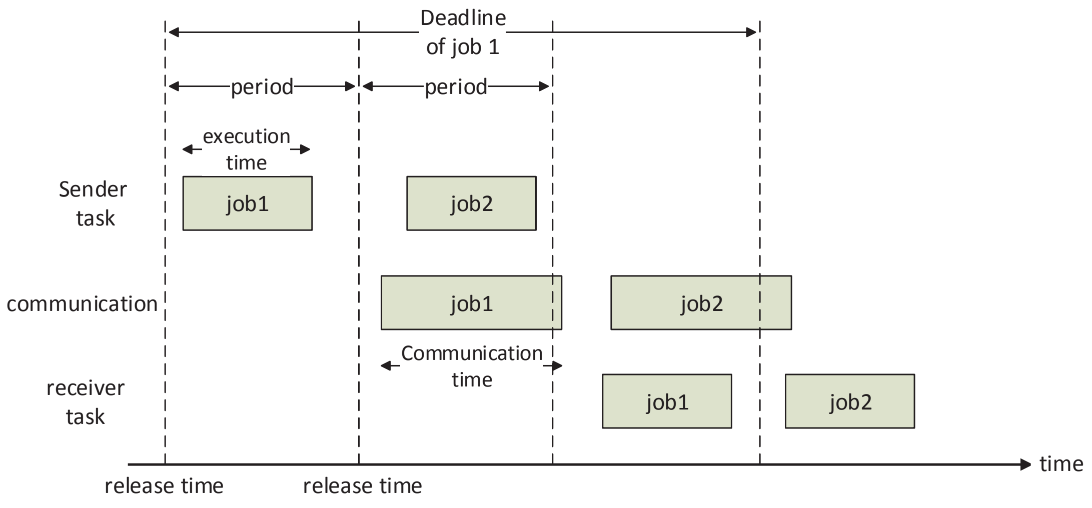


Fig. 2.2 Real-time parameters

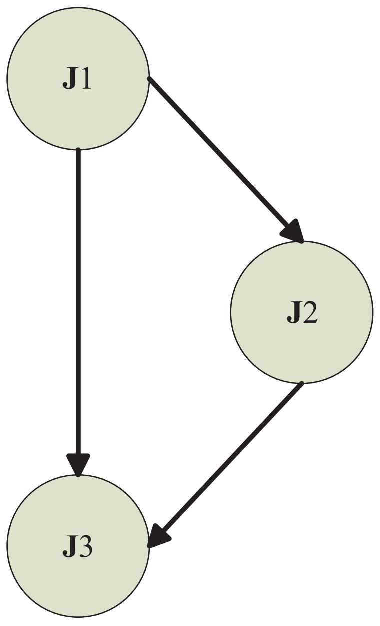


Fig. 2.3 Precedence relations among three jobs.

The precedence constraints among three jobs is illustrated as a DAG in figure 2.3. The Job J_1 is the only job that does not have predecessors and it can start execution first. The second job J_2 can start execution only when job J_1 is completed, whereas J_3 must wait for J_1 and J_2 to complete.

2.2.5 Real-time Scheduling

The solution of a scheduling problem of a set of tasks requires an ordered list according to which the tasks are to be executed where a number of constraints are satisfied. Throughout this thesis, the scheduling problems are represented by a set of jobs $J = j_1, j_2, \ldots, j_n$ and a set of End-systems (Processing machines) $P = p_1, p_2, \ldots, p_m$ that are used to process the jobs. The solution of the scheduling problem is called scheduling which is related to the assignment of End-systems from the set P to jobs from J so that all jobs are successfully completed under certain dependent constraints. In our work, it is assumed that each job is to be processed by at most one End-system at a time and each End-system can only process one job per unit of time.

The solution of the scheduling problem is to find out a schedule for the set of given jobs, where a message schedule is incorporated that guarantee the requirements of the planned real-time system by satisfying the reliability and predictability characteristics of the time-triggered network. A schedule need not to be only valid but feasible and in best cases optimal is given. A valid schedule for a set of jobs considers exclusive job assignments to End-systems, satisfying all precedence constraints, and guaranteeing inter-task communication by message transmission according to precedence dependencies. A valid schedule is said to be feasible only if the time constraints of the respective jobs are met; i.e., arrival and deadline of each job. An optimal schedule is the best feasible schedule according to some measures [Mal09]. For example, if the objective of a scheduling problem is to minimize the end-to-end latency of the overall jobs, the optimal schedule is the one that has the minimal end-to-end- latency.

2.3 Linear Programming

Linear Programming (LP) problems contain a set of unknown quantities to be optimized called decision variables. These variables are subject to certain requirements and restrictions by means of constraints. Each constraint requires that a linear function of the decision variable is either equal to, not less than, or not more than, a scalar value. The quality of the variables to be optimized is assessed by a linear function, which is called an objective function. The objective function is either a minimization or a maximization linear function. In MILP, some, but not all, variables are restricted to be integer [Baz92]. LP model is used to model different types of problems, such as routing, scheduling, and assignment. Various industries make use of LP models including transportation, telecommunications, and manufacturing.

In general, a linear programming problem can be represented as:

$$\begin{aligned}
\underset{x_1 x_2 \ldots x_n}{\text{minimize/maximize}} \quad & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\
\text{subject to} \quad & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \ \ (\leq, =, or \geq) \ \ b_1 \\
& a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \ \ (\leq, =, or \geq) \ \ b_2 \\
& \ldots \\
& a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \ \ (\leq, =, or \geq) \ \ b_m \\
& x_j \geq 0 \quad \forall j = 1, \ldots, n
\end{aligned}$$

Values $c_j, \forall j = 1, \ldots, n$, are the objective coefficients, and denote the costs associated with their corresponding decisions in minimization problems, or the profit generated from the corresponding decisions in maximization problems. The values $b_1, \ldots, b_m$ represent amounts of available resources (in case of $\leq$ constraints) or requirements (in case of $\geq$ constraints). The amount of resources consumed or requirements needed are typically represented by the values a_{ij}.

A solution in which all constraints are satisfied is called a feasible solution. An optimal solution is a feasible solution that gives the best objective function value. In case no solution exists to an MILP problem, the MILP problem is called infeasible.

MILP problem are generally solved using branch-and-bound algorithm in which the search space of the problem is divided into a sequence of subproblems. A MILP problem has a search space that can be represented by a tree where its root represents the original problem and its nodes are subproblems that are generally derived from the root.

The branch-and-bound algorithm starts with calculating the relaxation of the MILP problem which represents the optimal solution of the relaxed MILP problem using one of the standard linear programming techniques. The relaxation of a MILP is taking the same constraints and the objective function in the LP model while dropping the integrality requirements of the variables. If the optimal LP solution contains integer values for the variables, then the optimal solution of the MILP is also the optimal solution of the relaxed MILP problem. Otherwise, we need to perform a rounding procedure that transforms $\bar{x}_i$ into an integral value x_i. The procedure divides the problem into two subproblems, called *active* *nodes*, where each subproblem is identical to the original subproblem with a new constraint related to the *branching variable* x_i: $x_i \leq \lfloor \bar{x_i} \rfloor$ in the first subproblem and $x_i \geq \lceil \bar{x_i} \rceil$ [AS05].

After that, the algorithm chooses one of the active nodes and starts solving the LP relaxation of its subproblem. If the solution is non-integer feasible, then the algorithm defines two new subproblems similar to the predecessor divisions. If the subproblem can result to a solution that generates an integer, then the solution is checked for feasibility. Infeasible solution requires the algorithm to drop the current subproblem, i.e., the active node is *fathomed*. The objective value of a feasible solution provides an upper bound for the objective value in the MILP problem. This upper bound, in case of minimization, is used to drop active nodes that have lower bounds higher than an existing upper bound.

MILP problems are generally non-deterministic polynomial-time hard (NP-hard), in which the solution time needed to solve a MILP problem grows exponentially with the size problem size. The branch-and-bound algorithm has been improved to speed up the search progress where heuristics methods are implemented for improving the upper bounds by finding feasible solutions. Moreover, preprocessing technique reduces the problem size and enhances the problem solvability by tightening the linear relaxation [LG11]. Other improvement is called *cutting planes* technique that adds constraints to the subproblem which cut-off the optimal LP solution while keeping one or more optimal integer solution. Cutting planes technique tends to reduce the relaxed feasible region and in optimal cuts part of the MILP feasible region may be removed which results in lower bound improvement.

2.4 Dependability

One of the most important non-functional attributes of real-time systems is *dependability*. It is defined as a measure of the ability of a system to provide its agreed level of service to its users [Dub13]. Another broader definition is provided by the International Electrotechnical Commission (IEC) IEV 191-02-03: "dependability is the collective term used to describe the availability performance and its influencing factors: reliability performance, maintainability performance and maintenance support performance". According to [ALR$^+$01], the concept of dependability consists of three classifications: faults, means, and attributes of dependability.

2.4.1 Faults and Fault-Tolerance

Distributed systems are used to offer a dependable service to the users of the system. When the user does not receive the intended service, i.e., the system behavior deviates from its intended service, a system *failure* event exists. Failures happen due to an unintended state

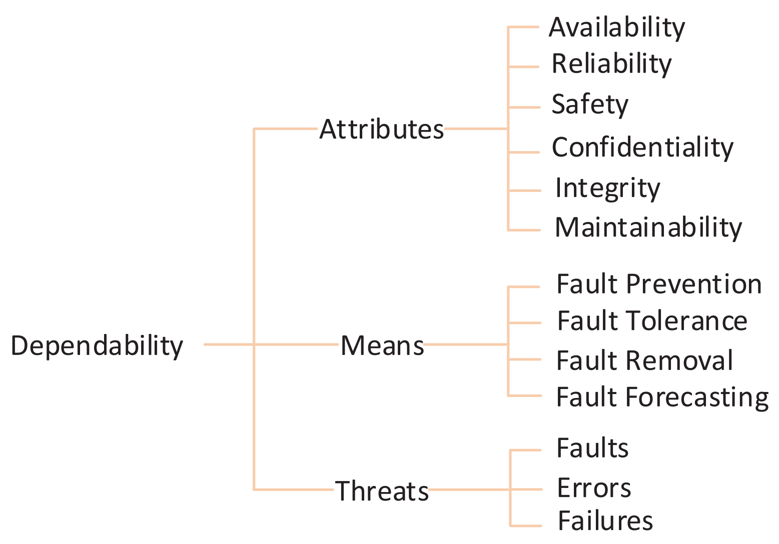


Fig. 2.4 Dependability Tree

within the system, which is called an *error*. An error is the result of an incorrect computation that may be the cause of a failure if not detected before altering a service by one of the error detection mechanisms. The main origin of errors are *faults*. A fault is an adverse phenomenon that can occur in the system such as a bit change in a memory or an uninitialized variable in software. A fault is active when it results in an error, otherwise it is dormant [Kop11] [ALR+01].

To better understand the above mentioned terms, consider a sensory input data, which contains information about the current wheel speed in an ABS system of an automobile. The DRAM memory storing this sensor data can encounter a bit flip of its cells, e.g., due to a rowhammer effect. Assuming the memory is not protected by a parity bit the system will produce an incorrect brake pressure value (computation error) which causes a failure that results in a longer stopping distance of the car. However, the faulty value is called dormant if the driver does not trigger the brake.

Faults can be classified into three main groups depending on duration, nature and scope [Nel90]. The fault's duration can be transient, intermittent, or permanent. A transient fault occurs at short duration and it is non-periodic. An intermittent fault is a sporadic occurrence of faults due to unstable component operation. A permanent fault remains a fault and is not corrected by time due to component failures or damage. The unintended system behavior determines the nature of a fault that may be logical or indeterminate. The error produced by determinate faults are easy to model since they can be represented as logical values compared to indeterminate faults which do not have a digital representation, such as the floating of a signal voltage between logic 1 and 0. The last classification of faults relates to the area that is affected at the considered level of abstraction. It is classified into two sub-categories: local faults that affect only one module and distributed faults that have effect on more than one module and may affect the entire system [Avi76].

2.4.2 Dependability Means

The means to attain dependability are grouped into four classes [ALRL04]. *Fault prevention* includes methods and techniques that are applied to eliminate the occurrence of faults. These techniques represent the quality control techniques that are employed during the design phase of hardware and software.

During system operation, faults can occur at any time in any part or component of the system and disrupt the delivery of correct services. *Fault tolerance* is responsible for keeping the system operational and providing reasonable services in the presence of faults. The implementation of fault tolerance consists of detecting errors followed by a system recovery mechanism that transforms the faulty state of the system into a state without errors. Fault-tolerant systems must be based on a clear understanding of the fault assumptions as specified by the a fault hypothesis which is covered in section 2.5.

Fault removal is a repeated process, which is performed in the development phase as well as in the maintenance phase, consisting of three main steps: verification, diagnosis, and correction. The first step concerns observing the system to detect any violation of the specified verification conditions. If such a deviation exists, a diagnosis process is triggered, which identifies the fault(s) that caused the occurrence of the detected violation. The last step concerns performing corrective actions to eliminate faults.

Fault forecasting is an evaluation and estimation of the possible future occurrences of faults with their consequences on the system behavior. It consists of a set of methods to evaluate the system behavior regarding faults occurrences and activations. The evaluation can be qualitative where failure types are identified and classified, or quantitative where a study of the satisfaction of the dependability attributes is based on a probabilistic evaluation.

2.4.3 Dependability Attributes

The expected properties of the system are called the dependability attributes. There are six major attributes of dependability: availability, reliability, safety, confidentiality, integrity, and maintainability. The importance of each one of these attributes depends on the intended application of the system.

Reliability is the continuity of correct service delivery, also called up-time. It is defined as the probability of the system to provide correct and continuous behavior during a period of system operation. In other words, reliabilityy is a function of time and the considered interval of time depends on the nature of the system being considered. For example, the mission time of a spacecraft is significantly longer than the mission time of an aircraft flight. Alternatively, unreliability is the probability of failure, which is the probability that the system fails to deliver correct services during a time period. Safety-critical systems, such as modern aircrafts, have failure rates in the order of 10^{-9} critical failures per hour [Bow00].

while critical failure of a system must be infinitly avoided, sn infinite failure-free operation of a component is impractical due to high cost [Nel90]. Systems subject to failures need to have low repair time in order to minimize downtime. *Availability* is the probability that the system is in an operational state and provides correct services when needed [Sho01]. It is measured by the fraction of time that the system is ready to provide correct services. Availability can be calculated based on the reliability (uptime) and the repair time (downtime) as follows:

$$A = \frac{\text{Uptime}}{\text{Uptime} + \text{Downtime}}$$

Compared to reliability that considers that all failures are equal, *safety* partitions failures into two main groups namely: fail-safe which is a noncritical failure and fail-unsafe which causes disastrous results. For example, an intrusion alarm system might malfunction in which it fails to function properly in case an unauthorized access occurs (fail-unsafe), or it may give a wrong alarm when no danger exists (fail-safe). Safety is the reliability with respect to critical failure types [Kop11]. In other words, safety is the probability of the system to avoid the occurrence of a catastrophic failure within a certain period of time. Safety-critical applications, in which a failure may have catastrophic results such as human injury, life loss, or environmental damage, demand high safety measures. Examples are trains, avionics, medical systems, and military systems.

Security is related to the *confidentiality* and *integrity* of information. It is defined as the ability of the system to prevent unauthorized access to data and avoid data modification by unapproved access or component/communication error [Gas88]. Generally, security aims to protect the system against attacks that aim to harm system component hardware, cause information loss /damage, and disrupt services provided by the system. Most research in the field of embedded systems focuses on protocol and communication issues in the context of reliability and safety, while security remains an open challenge in present day of embedded systems. The study presented by [WWP06] stated that the internal communication connecting the embedded system components inside modern vehicles is insecure against malicious attacks. Moreover, these internal communication networks may even encounter external security attacks when coupled with Bluetooth or car multimedia networks, i.e., Media Oriented System Transport (MOST) or GigaStar.

Maintainability is a measure of the time required to restore a system after a failure occurrence. It is defined as the probability that the system is repaired within an interval of time.

2.5 Fault Hypothesis

Safety-critical systems are designed based on fault-tolerant architectures. The key point for designing a fault-tolerant architecture is the specification of a fault hypothesis. The fault hypothesis states assumptions about the types and the rates of faults and how end-systems may fail [OP06].

A fault hypothesis divides the fault-space into two partitions, namely normal faults and rare faults. The former set contains faults that must be tolerated by the fault-tolerance mechanisms to bring the system back into a correct state. The latter set contains faults that are outside the fault hypothesis and are not covered by the provided fault-tolerance mechanisms. However, a never-give-up strategy is used to bring the system back (without assurance) to its correct state [Kop11].

The principle of fault tolerance in safety-critical systems is based on the assumptions of failure independence among redundant units [BCDV91]. A Fault Containment Region (FCR) is defined as a set of components that operate correctly regardless of any arbitrary logical or electrical fault outside the region [LH94]. According to another definition given by [Kop11], a FCR is a set of subsystems that share one or more common resources that can be affected by a single fault. The assumption of failure independence among these FCRs can be justified by using independent power supplies and electrically isolated interfaces that connect these FCRs.

Moreover, error propagation by sending erroneous messages from one FCR to another one must be prohibited in order to prevent fault occurrence in healthy FCRs and thus disrupting the failure independence assumption [Zur04]. Error propagation can be either in the time domain where the send/receive instants of a message are not in agreement with the agreed schedule, or in the value domain where a message contains incorrect data. Error detection in the time domain can be performed by the architecture whereas the responsibility of performing error detection in the value domain is typically within the host. To ensure that a faulty FCR does not impact the error detection mechanism, the latter must be in different FCRs than the message sender.

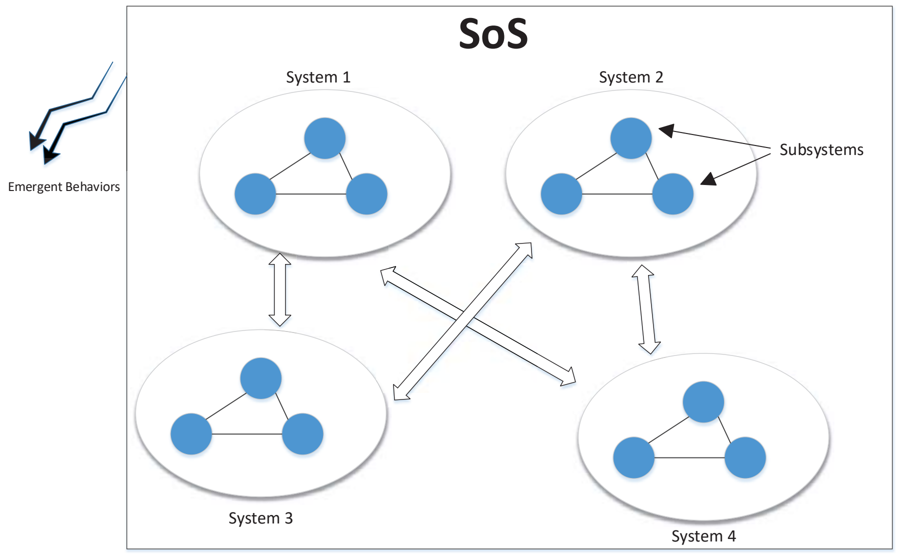


Fig. 2.5 System of Systems

2.6 Systems-of-Systems

A system is a set of interconnected components that interact in order to achieve a designed service which cannot be produced by components individually. These systems are basically managed by a central authority in which its behaviors are requested. When a number of these systems, each operated and managed independently, interact to fulfill an integrated overall goal, we speak of SoSs [Mai98]. This means that each constituent system in the SoS can still operate and produce useful results to fulfill its own purpose if separated from the SoS, while the main goal of coordinating these systems into an SoS is to deliver emergent services. Also, the management of each system is performed by its own when it is working independently as well as inside an SoS. The main reasons for the rising focus on SoS are the realization of different and emergent functionality that reach beyond the capability achieved by a single system, and the control of the complexity growth of large systems due to continuous evolution [Kop11].

There exist several definitions in the literature of SoSs that are based on the viewpoint of certain applications. [SJS07] defined SoSs as large-scale complex systems in which these systems are concurrent and distributed. Another definition by [ISO15] states that a "SoS brings together a set of systems for a task that none of the systems can accomplish on its own. Each constituent system keeps its own management, goals, and resources while coordinating within the SoS and adapting to meet SoS goals".

2.6.1 Characteristics of SoS

Autonomy is defined as "the ability to complete one's own goals within limits and without the control of another entity" [BS09]. An autonomous system is described by [SMG00] as a system that can operate and provide services independently and without any form of external help. Hence, managerial and operational independence are the main subsets of the autonomy property. In addition, an autonomous system must react to external stimuli in order to achieve the SoS purpose [Zei90].

Compared to the design of monolithic systems where the components can operate as an integral part of the whole system, each system in a SoS is considered as an autonomous system that can decide and choose to participate in a SoS. The choice to belong to a SoS and to accept cooperation with other systems to achieve higher goals is based on the own needs and benefits [GGS11]. Thus, the term *belonging* in a SoS refers to the acceptance ability and need to make a valued contribution to the goal of the larger entity [BS09].

In order for the systems belonging to the SoS to cooperate and achieve common goals, there should be some form of *connectivity* between them. The ability for a system to connect to a SoS and stay connected during the required time period to exchange data is a connectivity characteristic [BS09]. In other words, the connectivity property is the ability to achieve interoperability amongst the systems forming the SoS [BS06]. The connectivity determination for each system in an SoS, how and when interfaces and links are formed with other systems, is based on the system's needs.

Although autonomy refers to the capability of the connected systems to achieve a common goal, it does not explain the number of methods and capabilities available to perform such connections. The cooperation of these heterogeneous systems in an SoS requires a variety of capabilities in order to survive for a long time despite the introduction of different and emergent technologies. An SoS that has a variety of capabilities is called a *diverse* system and this property increases the overall capability of the SoS achieved by autonomy, belonging, and connectivity.

The behaviors and capabilities of a system are deliberately and intentionally designed. However, evolving behaviors and capabilities can be developed inside the SoS based on the other factors, such as autonomy of the related systems, belonging feature, enriched connectivity, and diverse behavior commitment of SoS [BS06]. Emergence is a term that refers to behaviors and services that result from the interaction of systems in an SoS and that are not inherent in any individual system. Emergence is defined by [Jam08] as "something unexpected in the collective behavior of an entity within its environment, not attributed to any sub-set of its parts, that is present (and observed) in a given view and not present (and observed) in any other view".

Chapter 3

Related Work

Distributed systems consist of a set of end-systems that share a communication network. Application's tasks compete for the shared physical resources in order to deliver a specific service of the system. Scheduling is the process of the temporal and spatial allocation of the shared resources for tasks and messages. This chapter gives an overview of related work in the area of scheduling algorithms.

3.1 Classification of Scheduling Algorithms

Task computations and message transmissions are the main operations that describe the functionality of real-time systems. These activities are described in terms of a precedence constraints graph according to which the operations need to be executed. The process of effectively assigning computation and communication activities to shared resources, such as CPU s and communication links, is the function of scheduling algorithms. According to [But11], scheduling algorithms can be classified as follows.

Static scheduling algorithms are those in which all decisions are based on fixed parameters, assigned to tasks before their activation [But11]. Thus, static scheduling requires a priori knowledge of all task attributes and it is considered less flexible. On the other hand, a dynamic scheduling algorithm is based on dynamic parameters that may change over time in the generation of scheduling decisions. This type of algorithm has the advantage of better resource utilization and support for unpredictable events, but its runtime is higher compared to static scheduling.

Scheduling algorithms are divided into two groups based on the time at which scheduling decisions are made. Offline scheduling algorithms generate scheduling decisions and store them in a dispatcher table before the system is deployed [But11]. The dispatcher table contains temporal properties of all participating tasks, such as the time instants at which each message is injected by the sending node till it is received by the received node. Offline scheduling is used to manage distributed applications with complex constraints where all system activities and properties are a priori known with deterministic time bounds. The second type is called on-line scheduling where scheduling decisions are determined at run-time. Generally, it is impossible to design an optimal on-line scheduling algorithm since less information and time are available, but an offline scheduling algorithm can guarantee deadlines where optimal ordering of tasks are found [Kop11].

Preemptive scheduling algorithms use the task priority characteristic in task scheduling where lower priority tasks can be suspended by tasks with higher priorities. This type of scheduling is more complex and requires more resources where poorly designed preemption strategies can lead to starvation of low priority tasks [GGLR98]. The priorities of tasks are determined randomly or based on the system design goal. The order of task selection is based on the priority given either statically or dynamically depending on their information availability. Non-preemptive scheduling algorithms are based on uninterrupted task executions [But11].

3.2 Scheduling of Distributed Systems

Due to the fact that embedded systems are used for specific purposes, they are different from general-purpose computers in terms of functionality. Consequently, embedded systems have restricted resources (e.g., power consumption, memory size, processing speed). In addition, embedded system users do not only expect its correct logical operation, but functions also need to be finished in a timely manner and with high reliability. Thus, Extra-Functional Properties (EFPs) such as reliability, energy-efficiency, and scalability have to be considered in the design of embedded system to address the trade-off between reliability and limited resources.

These EFPs are competing against each other. For example, a system that is designed for best performance can have high power consumption. Also, a system which is the best from the reliability view point tends to be the worse in terms of latency. As a consequence, one of the main problems in the design of embedded system is to find the optimal solutions for these EFPs in order to get the best property trade-offs. Multi-criteria optimization is the process of fine-tuning EFPs in order to get the best solutions possible. This is done by exploring the points in the Pareto curve [AHU], which is used to select the best property value under other property constraints. For example, the optimization challenge is to reduce the power consumption under a performance constraint which can be solved by finding the Pareto curve. However, the Pareto curve is found after a full search, where all points in the design space have to be explored. This search process tends to be unfeasible since it needs a long simulation time for function evaluation.

Numerous methods have been suggested to explore the system design space. The authors in [BAB96] proposed a toolset called SimpleScalar, which consists of a number of MIPS-based simulators for computing the influences of high level architectural trade-offs. It is specifically designed to find the architectures of processor and memory without considering the energy. SimplePower is suggested by [VKI$^+$00], based on SimpleScalar to evaluate the power of several aspects at the system-level. It computes the energy consumption considering several aspects of memory (I-cache, D-cache, address and data busses, and off-chip main memory) and the data path of the cores. An extension to the SimpleScalar simulator, called Wattch, is proposed by [BTM00] that integrates energy evaluation at the architectural level with a good level of accuracy for processor and memory subsystems. It is a low level estimation approach to evaluate energy consumption relating to performance trade-offs. It also supports finding different system optimization strategies to reduce power consumption. A framework called Avalanche, proposed by [LH98], explores the trade-offs between performance and power consumption for embedded systems in terms of software, memory and hardware.

In [GV02], the authors proposed an optimization structure, called Platune, to find estimations for Pareto curves without delving into the whole design space. This is specifically done by introducing the concept of parameter independence where each space can be analyzed separately which leads to short simulation time. However, the independencies of parameters needs to be defined by the user using a dependency graph and not by the framework itself. An extension to the Platune framework is designed by [PG02] which integrates genetic algorithm analysis for the optimization of dependent parameters. Similarly, the user needs to define the parameters independencies using a dependency graph. The work presented in [PSZ] exploits heuristic algorithms (i.e., Random Search Pareto (RSP), Pareto Simulating Annealing (PSA) and Pareto Reactive Tabu Search (PRTS)) to explore the design space and to find the Pareto curve for EFPs in short time. The analysis results showed that those algorithms minimize the time needed to get an approximate Pareto curve up to three orders compared to a full search. The Tri-criteria Scheduling Heuristic (TSH) is developed by [AGK12] and creates a static multiprocessor schedule as an output given a software application graph and a multiprocessor architecture as inputs. The produced schedule has three optimized characteristics: small schedule length, reliability and low power consumption. The length of the schedule is reduced by exploiting power efficient schedule pressure as a cost function. Its reliability is enhanced through the use of active replication of operations while the lower power consumption is accomplished by using dynamic voltage scaling.

The work done by [Ram90] presents the allocation problem as a global view, where the allocation algorithm aims to schedule periodic tasks to processors while taking into account precedence constraints. The proposed static algorithm assumes a TDMA protocol for a shared broadcast bus where all processors are connected to. Heuristics are used to allocate tasks to processors according to their communication overhead. Tasks which communicate with each other are moved to the same processor and preventing task allocations to non-schedulable processors.

[ACP04] explores the use of genetic algorithms to resolve the mapping problem. It finds an accurate approximation of the Pareto-optimal front of the mappings that minimize the amount of communication delay and the average power consumption. It is based on the SPEA2 algorithm [ZLT01] for mapping space exploration. The mapping solution is evaluated using a NoC simulator that shows the performance to be optimized and determines the fitness of the provided solution. The above two steps are iteratively executed till a stop criterion is met. The main drawback of this approach is the time required by the simulator to evaluate each solution where a high confidence on average delay and power consumption takes a long time by the simulator.

An automatic schedule generation algorithm for tasks and messages is introduced by [Far06]. The algorithm is based on the so-called Logical Execution Time (LET) abstraction, which abstracts from physical execution time and thereby from both the underlying execution platform and the communication topology. The idea of LET follows a timed-model; in which computational and communication activities are independent of the physical execution times and take a fixed amount of time. Thus, the temporal behaviors of a given program, such as task initialization and computational results, are determined previously and independent from the executing platform. This concept provides determinism and predictability where the program behavior depends only on the task property and the environment. The scheduling of tasks and messages is done in two steps. First, the messages are scheduled based on a heuristic algorithm called Latest Release Time (LRT). Then, the tasks are scheduled using Earliest Deadline First (EDF) with a precedence constraint algorithm. Task scheduling is based on the timing constraints of the corresponding messages. The messages are scheduled as late as possible in order to guarantee that producer task will finish before the corresponding message is sent on the bus. The LET concept introduces a so-called unit-delay, in which dependent tasks exchange information only at the LET boundaries and previous execution values remain in the outputs between LET.

An online scheduling algorithm for time-triggered messages is proposed by [MVPA13] that takes advantage of temporal redundancy to mask message errors due to transient faults. The proposed scheduling algorithm is in contrast to offline scheduling algorithms that always assume worse-case scenarios and consume bandwidth independently of the actual occurrence of errors. The proposed technique only consumes bandwidth when error occurs. The devised mechanism takes advantage of distinguishing features of the Flexible Time-Triggered (FTT) CAN protocol that has a built-in omission detection mechanism. In order to combine Event-Triggered (ET) and time-triggered communication with temporal isolation, the FTT-CAN protocol uses the dual-phase Elementary Cycle (EC) that is composed of two windows. Moreover, it relies on a periodic server to reschedule error messages which are then integrated into the time-triggered traffic. Although the implemented mechanism offers error recovery using much less extra bandwidth than typical techniques, it fails to reschedule messages that have short deadlines because the rescheduling process takes a considerable amount of time.

Other research generates a reliable time-triggered-message schedule called *(k, l)-resistant* schedule, based on the assumption of the degree of the link failures in the network [AGRN16]. A schedule is *(k, l)-resistant* if at least l messages are delivered by the global deadline when k edge crashes occur in the network. The problem was solved using an Satisfiability Modulo Theories (SMT)-based solver. It uses a *CEGAR-like* procedure to find a *(k, l)-resistant* schedule. First, it randomly selects the source and destination nodes for each message taking into account that the selected nodes are different. Then, it peaks an arbitrary message and finds the shortest path between the source and destination nodes by running Dijkstra's algorithm. After that, it computes second paths that represent the alternatives for the first path. This is done by having for each traversed node *v* on the first path, a second path that is the shortest path from v to the destination node while avoiding heavily loaded edges.

A simultaneous co-synthesis of network and application schedules with preemptive time-triggered tasks communicating in a switched time-triggered network is addressed in [CO14]. The scheduling constraints are formulated using SMT and solved using a SMT solver called Yices v2.2.1. The algorithm defines two types of tasks to be scheduled, communicating tasks that have dependencies and free tasks that do not require any network messages. An Incremental scheduling is performed where only communicating tasks are scheduled and then adding the free tasks to the end-system and checking whether the resulting model is schedulable in terms of end-system utilization. The communicating tasks are allocated to randomly selected end-systems. If the resulting schedule is unfeasible, the problem is backtracked and the increment size is increased by adding those free tasks that result in unfeasibility. The procedure is incrementally repeated until either a solution is found or all the free tasks have been added to the scheduling problem.

3.3 Research Gap in the State of the Art

A scheduling model for multi-core architectures based on network on chip as well as for distributed systems with support of real-time characteristics, fault-tolerance, and different timing models is missing in the state of the art. Previous research works outlined in section 3.2 either provide scheduling model for time-triggered communication without considering rate-constrained communication or their models do not support fault tolerance of communication links.

To the best of our knowledge, a scheduling model for SoS architectures that support real-time communication is still missing due to the lack of central knowledge and control. Also, the overall computation for the whole constituent systems of the SoS creates another issue that requires a huge memory size and processing time.

Moreover, a simulation framework for validating SoS models is not available in the state of the art. Because of its dynamic nature of interaction between the constituent systems, it is impossible to validate SoS scheduling model in real networks.

This dissertation advances the state of the art by introducing a scheduling model for a networked multi-core architecture that supports real-time communications as well as rate-constraints. Moreover, the proposed model finds optimal end-to-end paths of application communications while taking into account collision avoidance and precedence constraints of dependent jobs. Moreover, a fault tolerance technique is integrated into the introduced model to maximize the reliability of message delivery. This is done in the form of redundant messages where each message is sent through different paths in which their communication links have good reliability.

Furthermore, the proposed scheduling model is further extended to allow scheduling of SoS applications. A so called incremental, distributed, and concurrent scheduling model is proposed to schedule applications that need to be executed in a number of differently managed constituent systems.

Finally, a simulation framework for SoS consisting of the main building blocks (end-systems and switches) is proposed. This simulation framework is used to evaluate and validate the scenario results of the proposed SoS scheduling model.

Chapter 4

Scheduling and Allocation of Time-Triggered and Event-Triggered Services for Multi-Core Processors with Networks-on-a-Chip

This chapter proposes a scheduling model that supports different timing models and inter-job dependency for networked multi-core chips [MOAK15]. First, a conceptual model of the NoC architecture where its physical platform (in which network interfaces are connected to switches) as well as logical model (in which a number of jobs have precedence relationships in the form of message transmissions) are described. Then, an optimization model for the scheduling problem is formulated as a MILP model. Finally, an evaluation for the presented model is conducted and its results are discussed.

4.1 Network-on-Chip Architecture

As communication has become the bottleneck in many of today's digital systems, the interconnection is a dominant factor in determining performance and timeliness in real-time embedded systems [MB06]. In this section, we introduce the system model of a multi-core architecture for safety-relevant embedded systems.

4.1.1 Multi-Core Platform

The bus-based architectures have been substituted by NoCs in the past decades as today's Socket-on-Chips (SoCs) demand more and more interconnection capacity. A typical NoC is mainly composed of the *on-chip Network Interfaces* (NIs) and *on-chip switches*. An Network Interface (NI) serves as the interface to the NoC for the processing cores by injecting the messages from the cores into the NoC as well as delivering the received messages from theNoC to the cores. Switches are responsible to relay the flits – fractions of messages – from the sender's NI to the destination NIs. *Physical links* serve for the interconnection among NIs and switches.

We can distinguish between autonomous and source-controlled NoCs. In autonomous NoCs, the switches have their own configurations that determine the temporal control and the distributed routing of flits. In source-controlled networks in contrast, the information about the complete path through the network is injected alongside with the message. This routing paradigm is known as source-based routing [DYN03b].

An on-chip message consists of packets, each of which includes flits, typically starting with a head flit, successive data flits and a tail flit. The head flit, in case of source-based routing includes the complete path to the destination and in case of an autonomous network, the head flit will include the destination address. Each flit traverses a path on the NoC starting from the sender's NI, taking multiple hops via switches until reaching the destination NI.

4.1.2 Topology

The number of input and output units at each switch and the connection pattern of these units represent the *topology* of the NoC. We can distinguish between topology-dependent and topology-independent NoCs. While multicore SoCs tend to use topology-independent structures, a topology-dependent NoC offers a considerable degree of flexibility in arbitration, routing, flow control and queue size [M. 09].

There are several well-known topologies for NoCs, each of which offer beneficial properties in different applications. For instance, n-dimensional meshes or tori, are common examples of point-to-point NoC solutions with regularity, scalability, and conceptual simplicity. Despite a relatively large diameter and average distance, regular, low dimensional, point-to-point NoCs benefit from this cost-effective topology for regular use-cases. However, the large diameter of the mesh has a negative effect on the communication latency. Therefore, the Spidergon topology has been introduced [CLM+04] to enhance the network diameter and consequently to offer less communication latency by offering a simple bidirectional ring with extra cross links from each switch to its diagonally opposite neighbor. This topology has been employed by STNoC in order to offer cost and performance enhancements and a simple switch architecture [M. 09]. The ring topology on the other hand offers symmetry and a low node degree at the cost of a high average distance.

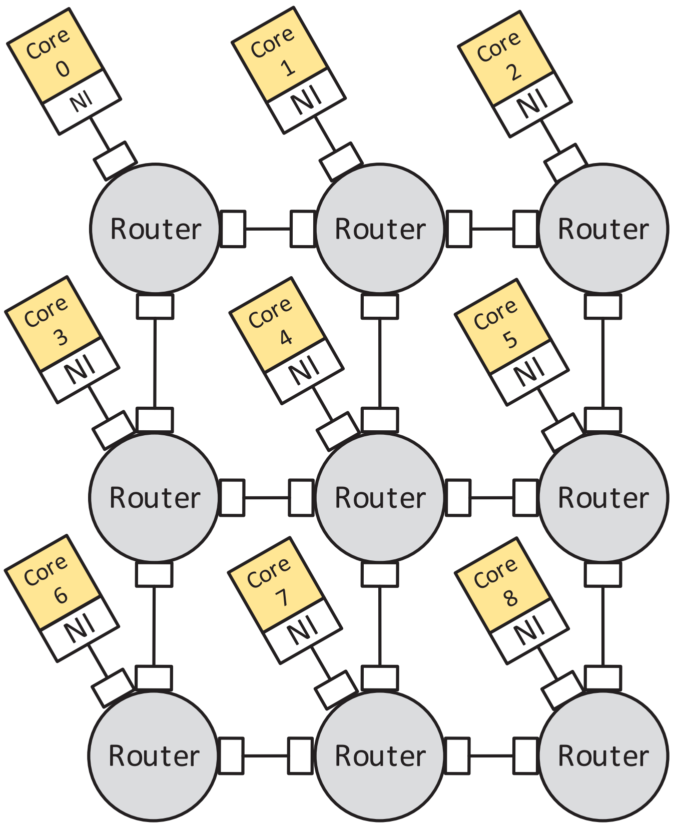


Fig. 4.1 Example NoC Topology (Mesh)

Figure 4.1 shows an example topology with a 2-dimensional $n \times n$ mesh where each core is connected to a single switch. This topology consists of n^2 cores arranged in a two-dimensional $n \times n$ grid.

4.1.3 Application and Jobs

An SoC serves as the platform for the realization of an embedded application (e.g., automotive control functions in an in-vehicle system). We model an application as a set of jobs, which depend on each other due to input/output relationships. Formally, the jobs are represented by a DAG.

In this thesis we assume that each core hosts at most one job. In this way, we establish a one-to-one mapping between the cores and the jobs. The communication between the jobs is performed via the NoC. Each job needs to wait for the information provided by other jobs (located on other cores). Likewise, it needs to deliver the processed data to other jobs via messages on the NoC. In case of time-triggered messages, the dependency means that a job needs to have the requested data ready before the deadline defined by the time-triggered schedule. This defines dependencies among jobs and consequently cores – due to the one-to-one mapping between jobs and cores.

Since the NIs add the routing information to the messages, cores do not need to be aware of the physical routes. Each message has a predefined path to the destination. Hence, we bind messages to paths represented as predefined links. This information is part of the configuration parameters at NIs. The time-triggered schedule as well as the temporal characteristics of event-triggered messages will also be stored at the NIs.

4.2 Scheduling Model

This section introduces a system model for a multi-core processor deployed with a NoC for time-triggered and event-triggered communication. We support both periodic communication with a fixed period and phase with respect to a global time base, as well as sporadic communication with rate-constraints.

The system model is mapped to a MILP problem for optimizing the allocation and scheduling of computational and communication activities. Arbitrary application graphs and NoC topologies are supported in order to permit the instantiation for different processor architectures and application-specific multi-core chips. Constraints ensure correct allocations based on a given multi-core architecture and application model (e.g., dependencies between communication and computational activities, limited connectivity between on-chip switches). The MILP problem also expresses real-time constraints and avoids collisions between time-triggered messages. Thereby, TDMA-based communication without dynamic arbitration is supported for time-triggered messages.

In the scheduling problem, two models are distinguished. A physical model describes the on-chip resources including the cores, the switches and their connectivity via the NoC. The second model is a logical model of the application that defines jobs and their dependencies based on exchanged messages. The term end-system will be used instead of core during the remaining parts of the thesis.

Table 4.1 summarizes the constants with the associated value domains. In the physical model, we consider a system with n nodes consisting of switches SW and end-systems ES. The nodes are connected using bi-directional physical communication links which can be described by a two-dimensional array C, in which the n^2 values of the matrix are either 0 (not connected) or 1 (connected).

Domain	Constant name	Type/size	Description
Physical Model	n	$\mathbb{N}$	Number of nodes
	C	$\begin{bmatrix} c_{1,1} & \cdots & c_{1,n} \\ \vdots & \ddots & \vdots \\ c_{n,1} & \cdots & c_{n,n} \end{bmatrix} \in \{0,1\}^{n\times n}$	Node Connectivity
	ALLOC	$[al_1 \ldots al_n]^T \in \{0,1\}^n$	Allocability
	U	$[u_1 \ldots u_m]^T \in \mathbb{N}^m$	H-to-H transmission time
Logical Model	j	$\mathbb{N}$	Number of jobs
	m	$\mathbb{N}$	Number of messages
	S	$[s_1 \ldots s_m]^T \in \{0,...,j-1\}^m$	Sender jobs
	D	$\begin{bmatrix} d_{1,1} & \cdots & d_{1,j} \\ \vdots & \ddots & \vdots \\ d_{m,1} & \cdots & d_{m,j} \end{bmatrix} \in \{0,1\}^{m\times j}$	Destination jobs
	T	$[t_1 \ldots t_m]^T \in \mathbb{N}^m$	Interarrival time
	E	$[e_1 \ldots e_j]^T \in \mathbb{N}^j$	Job' execution time
	MT	$[mt_1 \ldots mt_m]^T \in \{0,1\}^m$	Message type

Table 4.1 Overview table with constants

The logical model comprises a set of j jobs and a set of m messages to be exchanged. The vector S denotes the sender job of each message. Each job can send one or more messages, while a message is sent to only one receiving job. The input/output relationships between jobs as imposed by the messages are captured using a two-dimensional array D. For example, $d_{2,3} = 1$ denotes that message 2 is transmitted to job 3.

Depending on the timing model, either a period and phase of a time-triggered message or a minimum interarrival time of an event-triggered message is expressed by the timing parameter T. To differentiate between time-triggered and event-triggered messages, a message type is expressed in the boolean array MT where the value of one is set for the time-triggered messages while event-triggered messages are set to zero.

Each message requires a certain time, depending on the size of the message, to be transmitted on one link (from one hop to another hop). Thus, every time a message is sent from one hop to another, its time is advanced by a Hop-to-Hop (H-to-H) transmission time U. The computational time of jobs E is the execution time needed by the receiving job before sending a subsequent message.

In order for jobs to be allocated to only end-systems, allocability data is defined by $ALLOC$ where a value of 1 is set for end-systems and 0 for switches. The maximum number of hops a message can travel between the source and the destination is found by counting the number of switches SW.

4.2.1 Decision Variables

4.2.1.1 Job Allocation

These variables are used to denote the allocation of jobs to nodes of the physical platform model. The maximum value is determined by the number of nodes n.

$$A = \begin{bmatrix} a_1 \\ \vdots \\ a_j \end{bmatrix} \in \{1, ..., n\}^j$$

To ensure that each job is allocated to exactly one end-system, a boolean matrix $ALLOCM$ is used where the rows relate to jobs and columns to nodes. For example, $mat_{3,1} = 1$ means that job 3 is allocated to node 1.

$$ALLOCM = \begin{bmatrix} mat_{1,1} & \dots & mat_{1,n} \\ \vdots & \ddots & \vdots \\ mat_{j,1} & \dots & mat_{j,n} \end{bmatrix} \in \{0,1\}^{j \times n}$$

4.2.1.2 Hop Count

A message is injected at the source end-system, where the sender job was allocated. The message needs to be transported along one or more switches to the end-system of the destination job. In order to express the number of visited switches for each message, the vector hop count H is used and the maximum value of its elements denotes the critical path length. In the absence of cyclic paths, the maximum path length is $Max_H = SW + 1$.

$$H = \begin{bmatrix} h_1 \\ \vdots \\ h_m \end{bmatrix} \in \{1, ..., Max_H\}^m$$

4.2.1.3 Injection Time

This one-dimensional array represents the times by which the messages are injected in the network-on-a-chip.

$$I = \begin{bmatrix} i_1 \\ \vdots \\ i_m \end{bmatrix} \in \{N\}^m$$

4.2.1.4 Path and Visited Nodes

To record the path between the message's source and destination end-system, the path array P is used. Each row represents the path of a message starting from the node which allocates a source job to the node in which the destination job is allocated. For example, $p_{1,3} = 5$ means that the third node in which message number 1 visits is node number 5. The maximum number of nodes in a path equals the maximum number of hops along with the source end-system.

$$P = \begin{bmatrix} p_{1,1} & \cdots & p_{1,z} \\ \vdots & \ddots & \vdots \\ p_{m,1} & \cdots & p_{m,z} \end{bmatrix} \in \{1,...,n\}^{m \times z}$$

where $z = Max_H + 1$.

For the purpose of calculating the end-to-end latency, a matrix O is used to denote the switches that are passed by a message. For example, $o_{2,3} = 1$ means that message 2 meets node 3.

$$O = \begin{bmatrix} o_{1,1} & \cdots & o_{1,n} \\ \vdots & \ddots & \vdots \\ o_{m,1} & \cdots & o_{m \times n} \end{bmatrix} \in \{0,1\}^{m \times n}$$

4.2.2 Scheduling Constraints

This part describes the constraints that are used in the scheduling of time-triggered and event-triggered messages.

4.2.2.1 Connectivity Constraint

The first constraint considers the path topology of the network based on the node connectivity C. If there is no direct connection between two nodes a and b, then the path of a message must not include a hop from a to b.

$$\begin{aligned}
&\forall m_1 \in \{1,...,m\}, \forall a \in \{1,...,n\}, \forall B \in \{a+1,...,n\},\\
&\forall r \in \{1,...,Max_H\}:\\
&c_{a,b} = 0 \rightarrow (r+1 > h_{m_1})\\
&\vee ((a \neq p_{m_1,r} \vee b \neq p_{m_1,r+1}) \wedge (b \neq p_{m_1,r} \vee a \neq p_{m_1,r+1}))
\end{aligned} \tag{4.1}$$

4.2.2.2 Collision Avoidance Constraint

To ensure that no collision occurs, the scheduling of time-triggered messages ensures that no two messages are transmitted in one link at the same time. Thus, the messages should be transmitted in different paths or one needs to be scheduled before or after the transmission of the other one.

$$\begin{aligned}
&\forall m_1 \in \{1,...,m\}, m_2 \in \{m_1+1,...,m\},\\
&\forall r_1, r_2 \in \{1,...,Max_H\}:\\
&mt_{m_1} = 1 \wedge mt_{m_2} = 1\\
&\rightarrow (p_{m_1,r_1} \neq p_{m_2,r_2} \vee p_{m_1,r_1+1} \neq p_{m_2,r_2+1}\\
&\quad \vee r_1+1 > h_{m_1} \vee r_2+1 > h_{m_2}\\
&\quad \vee i_{m_1} + (r_1+1) \cdot u_{m_1} < i_{m_2} + r_2 \cdot u_{m_2}\\
&\quad \vee i_{m_2} + (r_2+1) \cdot u_{m_2} < i_{m_1} + r_1 \cdot u_{m_1})
\end{aligned} \tag{4.2}$$

4.2.2.3 Job Dependency Constraint

Depending on the precedence constraints between the jobs, jobs may need to wait for the output of the transmission of other jobs before they begin transmission. This constraint ensures that if a job sends a message m_1 to another job that needs the output of m_1 in order to send m_2, the start time of m_2 must be after the end of the transmission and execution of m_1.

$$\begin{aligned}
&\forall m_1, m_2 \in \{1,...,m\}, \forall j_1 \in \{1,...,j\}:\\
&d_{m_1,j_1} = 1 \wedge s_{m_2} = j_1\\
&\rightarrow i_{m_1} + h_{m_1} \cdot u_{m_1} + e_{j_1} < i_{m_2})
\end{aligned} \tag{4.3}$$

Each message must reach the destination node within its path and the selected number of hops.

$$\begin{aligned}
&\forall m_1, m_2 \in \{1,...,m\}, \forall j_1 \in \{1,...,j\}, \forall r_1 \in \{1,...,z\}:\\
&d_{m_1,j_1} = 1\\
&\rightarrow (p_{m_1,r_1} = a_{j_1} \wedge r_1 < h_{m_1})
\end{aligned} \tag{4.4}$$

Where $z = Max_H + 1$.

4.2.2.4 Job Assignment Constraints

These constraints ensure that each job is assigned to exactly one node (end-system). This is done by having the sum of each row in ALLOCM, i.e., each job, equal to 1. The allocated end-systems are stored in the allocation array A.

$$\begin{aligned} &\forall j_1 \in \{1,...,j\}: \\ &\quad \left(\sum_{r_1=1}^{n} mat_{j_1,r_1}\right) = 1 \\ &\quad \left(\bigvee_{r_1=1}^{ES} (mat_{j_1,r_1} = 1 \wedge a_{j_1} = r_1)\right) \end{aligned} \tag{4.5}$$

For jobs to be assigned only to end-systems and not to switches, the following constraints check the allocability constant (ALLOC), where nodes with a value of 0 must not be allocated jobs. This is done by requiring the sum of the switches columns in the ALLOCM matrix to be zero. To allow only one job to be allocated to an end-system, the sum for each end-system must be less than or equal to one.

$$\begin{aligned} &\forall r_1 \in \{1,...,n\}: \\ &\quad al_{r_1} = 1 \rightarrow \left(\sum_{j_1=1}^{j} mat_{j_1,r_1}\right) \leq 1 \\ &\quad al_{r_1} = 0 \rightarrow \left(\sum_{j_1=1}^{j} mat_{j_1,r_1}\right) = 0 \end{aligned} \tag{4.6}$$

In order to start the path of each message with the end-system that was allocated the job, the first node for each message path $p_{1,1} \dots p_{m,1}$ is set to the allocated node.

$$\begin{aligned} &\forall m_1 \in \{1,...,m\}, \forall j_1 \in \{1,...,j\}: \\ &\quad s_{m_1} = j_1 \\ &\quad \rightarrow \left(\bigvee_{r_1=1}^{ES} (a_{j_1} = r_1 \wedge p_{m_1,1} = r_1\right) \end{aligned} \tag{4.7}$$

4.2.2.5 Message Deadline Constraints

These constraints define the restrictions for the end-to-end message durations in order to ensure the temporal delivery of strictly periodic behavior of time-triggered messages as well as the less stringent rate-constrained messages. This is done by evaluating the end-to-end latency in addition to the execution time in the message sink job where the result must not exceed the message period.

$$\begin{aligned}&\forall m_1 \in \{1,...,m\}:\\&\quad \forall j_1 \in \{1,...,j\}\\&\qquad d_{m_1,j_1} = 1\\&\qquad\quad \rightarrow i_{m_1} + h_{m_1} \cdot u_{m_1} + e_{j_1} < t_{m_1})\end{aligned} \tag{4.8}$$

4.2.2.6 Bandwidth Constraints

The bandwidth constraints avoid discarding of messages due to insufficient bandwidth and buffer capacity of switches. First we need to express the visitation of nodes by messages: $o_{x,y}$.

$$\begin{aligned}&\forall r_1 \in \{1,...,Max_H\}, \forall m_1 \in \{1,...,m\}, \forall r_2 \in \{1,...,n\}:\\&(r_2 = p_{m,r_1} \wedge r_1 < h_{m_1})\\&\rightarrow o_{m_1,r_2} = 1\end{aligned} \tag{4.9}$$

$$\begin{aligned}&\forall m_1 \in \{1,...,m\}, \forall r_1 \in \{1,...,n\}:\\&\left(\bigvee_{r_2=1}^{h_m} (r_1 \neq p_{m_1,r_2})\right)\\&\rightarrow o_{m_1,r_1} = 0\end{aligned} \tag{4.10}$$

Then, the bandwidth at each node is determined by summing up the utilization (i.e., ratio of transmission time and minimum interarrival time) imposed by each message visiting the node:

$$\forall r_1 \in \{1,...,n\}: \left(\sum_{m_1=1}^{m} \frac{u_{m_1}}{t_{m_1}} \cdot o_{m_1,r_1} < 1\right) \tag{4.11}$$

4.2.2.7 Other Constraints

A value of 0 is required for all elements of the path after the hop count in order to ensure that the path finishes at the destination node.

$$\begin{aligned} &\forall m_1 \in \{1,...,m\}, \forall r_1 \in \{1,...,z\}: \\ &r_1 > h_{m_1} \\ &\rightarrow p_{m_1,r_1} = 0 \end{aligned} \tag{4.12}$$

where $z = Max_H + 1$.

In order to reduce scheduling time, loops should be avoided in each message such that a node is not visited more than once in the search process.

$$\begin{aligned} &\forall m_1 \in \{1,...,m\}, \forall r_1 \in \{1,...,Max_H\}: \\ &\left(\bigvee_{r_2=r_1+1}^{h_{m_1}} (p_{m_1,r_1} \neq p_{m_1,r_2}) \right) \end{aligned} \tag{4.13}$$

4.2.3 Objective Function

The objective is to minimize the maximum transmission time over the time-triggered messages (i.e., minimization of critical path). This is done by first finding the transmission times of each time-triggered message, expressed as the sum of the injection time i_m and the number of hops h_m multiplied by the transmission duration of a message u_m. Then, the objective function minimizes the highest value among all these messages.

$$\begin{aligned} &\forall m_1 \in \{1,...,m\}: \\ &\quad mt[m_1] = 1 \rightarrow CP[m_1] = (i_{m_1} + h_{m_1} \cdot u_{m_1}) \\ &minimize \quad max(CP) \end{aligned} \tag{4.14}$$

4.3 Graph Generation

The scenarios generated in the analysis of the work in this thesis are based on the Stanford Network Analysis Platform (SNAP) library which is widely used in numerous academic researches [Les]. The SNAP is extended to enable the creation of physical and logical graphs in terms of undirected and directed graphs, respectively. The following is a brief introduction of the SNAP library.

4.3.1 Stanford Network Analysis Platform (SNAP)

The SNAP library has the ability to modify graph structures and to provide fast execution of graph algorithm. It provides its efficient operations in adding, deleting, editing nodes and edges in graph while having a limited overhead on graph algorithms. It is used in the analysis of large graphs since it requires a smaller amount of RAM than alternative representations. SNAP is provided as an open-source library in C++ as well as in python for major operating systems [LS16]. .

The SNAP library defines *graph* as a set of nodes and edges where each edge connects two nodes. Two types of edges can be drawn, directed and undirected. It can generate and analyze large networks with hundreds of millions of nodes and edges. The SNAP can generate regular as well as random graphs with detailed attributes and metadata on each node and edge.

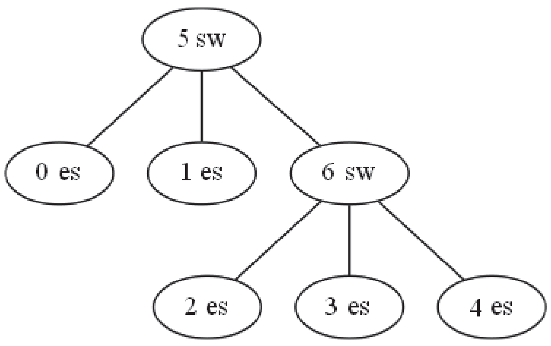
SNAP is based on fundamental classes, called graph and network containers, that are used to provide several types of graphs and networks. The generated graphs and networks can be directed and undirected graphs with the ability to create multi-graphs. Each node and edge connecting two nodes can have several attributes that define its characteristics in terms of color, position, and time. The graph and network containers can be accessed using a unified interface that is implemented with a number of generic methods. These methods are used to generate, manipulate, and analyze specific graph statistics.

Each node in the generated graph has a unique non-negative integer number called unique identifiers. These unique identifies do not have to be sequentially ordered from one to the number of nodes, but arbitrary integers. Edges of multi-graphs have a unique identifiers similar to the nodes. However, edges of simple graphs have no unique identifiers and can be accessed by providing the IDs of the pair of nodes that connect the edge.

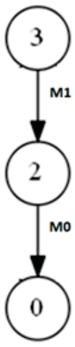
The nodes in a graph are represented by a hash table. In case of undirected graphs, the hash table stores the unique identifiers of the graph nodes where each node is associated with a vector of adjacent nodes connected to it. In case of directed graphs, the hash table contains a list of node unique identifiers and each node is associated with an outgoing nodes vector and an incoming nodes vector.

4.4 Results

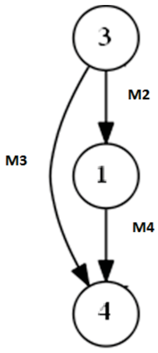
This section discusses the results of the MILP model described in 4.1. The main focus is to perform the scheduling and allocation of jobs and messages, while minimizing the critical path delay. In addition, we evaluate the computational time required to compute the schedule in CPLEX.



(a) Physical Model



(b) TT Logical Model



(c) ET Logical Model

Fig. 4.2 Model example with 7 Nodes, 5 Jobs and 5 msgs

Each example scenario consists of the constants explained in Table 4.1 in which both the physical and the logical models are defined. Consider an on-chip network with 7 nodes, 2 switches and 5 end-systems, where 5 jobs communicate and need to send 5 messages. The corresponding physical and logical models are shown in Figure 4.2, where figure a depicts the physical connection among nodes with bi-directional links and figures b and c depict the time-triggered and event-triggered logical models. The nodes represent the jobs and the arrows represent the messages sent from one job to another job. It should be noted that the first two messages are time-triggered while the other messages are event-triggered.

Table 4.2 shows the CPLEX input constants for the model in Figure 4.2 according to Section 4.1. The first constant describes the network model $(7,5,5)$ in which the model consists of 7 nodes and 5 jobs sending 5 messages. The second constant defines the node connectivity in the network C with bi-directional links. For example, the first node is connected to node 5 and the last node is connected to nodes $2,3,4$ and 5. Constant 3 shows the jobs that send the messages, where each job can send more than one message. For example, job 1 is the sender of the fifth message and job 3 is the sender of the messages 1, 2 and 3.

The fourth constant describes the receiving jobs of messages. For example, job 0 and job 2 are the receivers of the time-triggered messages 0 and 1 respectively, whereas job 4 is the receiver of event-triggered messages 3 and 4. Then, periods of time-triggered messages as well as the BAG for rate-constrained messages are described. The hop-to-hop transmission time U is defined in constant 6. For simplicity and better understanding of the example, all times of U are set to a constant with the value of 3 μs. Constant 7 describes the job execution times E and also here we set them all to a constant value of 2 μs.

The constant $ALLOC$ defines the ability of nodes to be allocated jobs where switches (i.e., the last two nodes) cannot be allocate jobs, while end-systems (i.e., nodes 0,1,2,3 and 4) can allocate jobs at a maximum of one job per end-system.

The last constant differentiates the type of the message to be transmitted. The value of 1 is set for time-triggered messages and 0 is used for event-triggered messages. In the examples, the first two messages are set to be time-triggered messages and the other messages are set to be event-triggered.

As illustrated in the previous section, the solution of the scheduling problem in on-chip time-triggered networks requires a large number of constraints. This makes the search space quite large for realistic network topologies which is illustrated in this section. The times of CPU calculation were obtained with CPLEX 12.6.1 running on a 12 processor Intel(R) Xeon(R), 2.2 GHz server with the operating system Linux Ubuntu 14.04.1.

Table 4.3 depicts the results of the MILP model solved by CPLEX. It gave an optimal solution for the problem within 5.99 seconds in which 498 constraints are evaluated. Moreover, the output presented the values of the decision variables described in Section 4.2.1. The allocation of jobs to end-systems is done in such a way that it minimizes the maximum transmission time of the time-triggered messages. Here, job 3, which is allocated to node 4, sends one time-triggered message to job 2 which is allocated to node 2, and two event-triggered messages to jobs 1 and 4 which are allocated to end-systems 1 and 0, respectively. The message with VL 1 is initiated at time 0 in end-system 4 and passes through switch 6 and is then received by end-system 2 with a total of two hops. Since only the first two messages are time-triggered, their scheduling is calculated in such a way that their transmissions are done without collisions. The last three messages can be scheduled without taking into account the collision constraints, which can be noticed in messages 2 and 3. They are scheduled to pass through the links between nodes 4, 6 and 5 and it is the job of the switches to resolve the contention between these event-triggered messages. Moreover, there is a dependency between job 2 and job 3 in the time-triggered messages and between job 3 and job 1 in the event-triggered messages. In the case of time-triggered messages, the transmission of message of VL 0 in job 2, allocated to end-system 2, starts at the end of the transmission of

Constant-No.	Constant Name	Data
1	n, j, m	[7,5,5]
2	C	[[0,0,0,0,0,1,0], [0,0,0,0,0,1,0], [0,0,0,0,0,0,1], [0,0,0,0,0,0,1], [0,0,0,0,0,0,1], [1,1,0,0,0,0,1], [0,0,1,1,1,1,0]]
3	S	[2,3,3,3,1]
4	D	[[1,0,0,0,0], [0,0,1,0,0], [0,1,0,0,0], [0,0,0,0,1], [0,0,0,0,1]]
5	T/BAG	[20,20,30,30,30]
6	U	[3,3,3,3,3]
7	E	[2,2,2,2,2]
8	ALLOC	[1,1,1,1,1,0,0]
9	MT	[1,1,0,0,0]

Table 4.2 CPLEX Input constants for model in Figure 4.2

message 1 in job 3 allocated to node 4. Thus, the injection time of message of VL 0 should be after the reception time of message of VL 1 and the execution time of this message in job 2 as shown in the following equation:

$$\begin{aligned} i_{VL0} &\rightarrow (i_1 + h_1 \cdot u_1 + e_2) \\ &\rightarrow (0 + 2 \cdot 3 + 2) \\ &\rightarrow 8\mu s \end{aligned}$$

Moreover, the temporal constraints for time-triggered messages are preserved in which all of these messages finish before their deadlines.

Job-No.	Alloc. Node	VL	Msg. Type	Period BAG	Path	Start Time	Finish Time
0	3						
1	1	4	ET	30	1-5-0	14	-
2	2	0	TT	20	2-6-3	8	14
3	4	1	TT	20	4-6-2	0	6
		2	ET	30	4-6-5-1	3	-
		3	ET	30	4-6-5-0	6	-
4	0						

Table 4.3 Results for model example

$$\begin{aligned} r_{VL0} &\rightarrow (i_0 + h_0 \cdot u_0 + e_3) < T_0 \\ &\rightarrow (8 + 2 \cdot 3 + 2) \\ &\rightarrow 16\mu s < 30\mu s \end{aligned}$$

Table 4.4 shows the execution times for different physical and logical topologies. It is evident that the solution time of the problem depends on the number of constraints which is affected by the number of nodes, jobs, and messages.

The proposed model in this chapter provides mapping of applications that have inter-job dependencies as well as supports different traffic classes such as time-triggered for periodic events and rate-constrained for sporadic events. The provided functions in the proposed scheduling model are of main requirements for mixed criticality systems. The mapping of jobs to end-systems and scheduling messages to communication paths are selected to obtain minimum overall end-to-end message latencies.

Moreover, the proposed model obtains an optimal solution to the scheduling problem that results in a guaranteed best minimization of end-to-end latency for periodic time-triggered messages. On the other hand, the injection times of rate-constrained messages are ensured to be sufficiently scheduled on the bandwidth gap left after time-triggered messages are scheduled.

Physical Model		Logical Model		No. of	Optimal?	Time
Sw	Es	Job	Msgs	Constraints		(s)
2	4	3	2	175	Yes	1.04
2	5	3	2	218	Yes	1.66
2	6	3	2	267	Yes	2.58
2	4	3	3	245	Yes	1.63
2	5	3	3	307	Yes	2.77
2	4	4	3	251	Yes	1.74
2	5	4	3	313	Yes	3.24
2	4	4	4	320	Yes	2.05
2	5	4	4	402	Yes	4.28
2	4	4	5	409	Yes	2.37
2	5	4	5	492	Yes	4.16
2	6	4	5	609	Yes	13.50
2	5	5	4	408	Yes	4.28
2	5	5	5	498	Yes	5.99
2	6	5	5	616	Yes	9.27
2	5	4	6	583	Yes	9.26
2	5	5	6	590	Yes	9.75
2	5	5	7	681	Yes	12.17
2	5	5	8	770	Yes	9.85
2	6	6	5	622	Yes	22.22
3	6	4	4	776	Yes	212.90
3	7	4	3	703	Yes	150.98

Table 4.4 Results of 22 case studies.

Chapter 5

Optimized and Reliable Scheduling Algorithm

This chapter introduces an improved scheduling model for a multi-core processors as well as for distributed systems with time-triggered and event-triggered communication. The new model reduces the search space of the scheduling problem and provides optimal solution with less time compared to the previous one. Moreover, reliability is integrated into the scheduling model in the form of redundant messages to mask link failures of time-triggered messages [MO17].

5.1 Improved Scheduling Model for Time-Triggered and Event-Triggered Messages

The solution of an MILP problem using a branch-and-bound algorithm requires to list all possible integer combinations of the decision variables and then selects the best feasible point. Such explicit enumeration is exhaustive and becomes impossible when evaluating a large number of variables. As a result, a number of enhancements to the branch-and-bound algorithm have been designed, such as preprocessing, heuristics, and cutting planes as described in section 2.3. These methods tend to implicitly consider all integer variable combinations for the general problem without necessarily evaluating them thoroughly. These implicit enumeration methods tend to reduce the search space of the problem to be solved and they are already integrated into most MILP solvers [AS05] [SAS].

Another way of reducing the problem search space is to add user-defined constraints that optimize the model to be solved. These user-defined constraints should not oppose the main

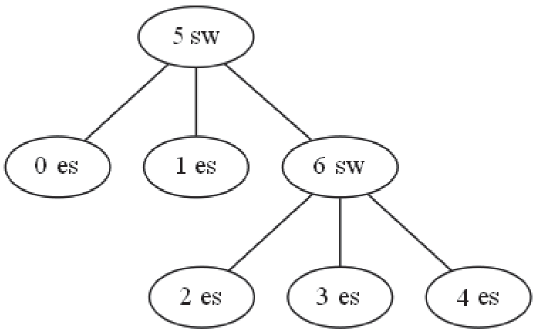


Fig. 5.1 Physical Model of 5 end-systems and 2 switches.

constraints which define the model but need to tighten the variables to be evaluated in order to reduce the time to obtain the optimal solution.

Consider the physical model in figure 5.1 which was used in the scenario model in the previous chapter. The allocation of jobs is only allowed on end-systems but not switches. Moreover, each end-system is connected to only one switch. This type of connectivity allows the scheduling model to consider only the network between switches while neglecting the end-systems.

Thus, the improved model provides the following enhancements.

- The search space of nodes allocability is reduced to include only end-systems. This restriction tightens the decision variables as well as the constraints associated with the allocation of jobs to end-systems.

- The end-to-end path calculation for each message will include only the path from the switch connected to the source's end-system to the switch connected to the destination's end-system. We call the switch that is connected to an end-system as a neighbor switch, listed in a vector D_{Switch}, which is determined by the connectivity matrix C. The introduction of neighbor switch to the previous model results in a massive reduction of the search space in the path decision variables as well as their related constraints.

5.1.1 Optimized Variables

The previously described model in section 4.2.1 defines a number of decision variables to be optimized according to the objective function. The most important variables to be optimized are the job allocation and the message path.

5.1.1.1 Job Allocation

These variables are used to denote the allocation of jobs to nodes of the physical platform model. Jobs cannot be allocated to switches, but only to end-systems. Thus, the maximum value is determined by the number of end-systems ES.

$$A = \begin{bmatrix} a_1 \\ \vdots \\ a_j \end{bmatrix} \in \{1, ..., ES\}^j$$

Also, the boolean matrix $ALLOCM$ is optimized to have a length of the number of end-systems ES instead of n.

$$ALLOCM = \begin{bmatrix} mat_{1,1} & \dots & mat_{1,ES} \\ \vdots & \ddots & \vdots \\ mat_{j,1} & \dots & mat_{j,ES} \end{bmatrix} \in \{0,1\}^{j \times ES}$$

To keep track of switches via which a job can transmit a message we use a vector SR. The vector SR denotes for each job a neighbor switch that is directly accessible from the end-system where the job is located. All other switches can only be reached by more than one hop. For example, $sr_2 = 14$ denotes that the switch with ID 14 is the neighbor switch for the end-system hosting the job 2.

$$SR = [sr_1 \dots sr_J]^T \in \{Z, ..., n\}^J$$

where $Z = ES + 1$.

5.1.1.2 Message Path

The rows in the path array P will now record the paths where each path starts from the neighbor switch of the end-system that allocates the source job to the neighbor switch of the end-system which allocates a sink job. The maximum value of these array elements is the maximum number of switches SW.

$$P = \begin{bmatrix} p_{1,1} & \cdots & p_{1,z} \\ \vdots & \ddots & \vdots \\ p_{m,1} & \cdots & p_{m,z} \end{bmatrix} \in \{1, ..., SW\}^{m \times z}$$

where $z = Max_H$.

5.1.2 Optimized Constraints

After optimizing the decision variable, the constraints related to these changed variables needs to be altered. These constraints are the connectivity, collision avoidance, and job assignment constraints.

5.1.2.1 Job Assignment Constraints

These constraints assign jobs to end-systems. A job must be allocated to exactly one end-system and each end-system can process at most one job. The $ALLOC$ matrix is used to allow the aforementioned restrictions where its rows represent jobs to be allocated and its columns denote the available end-systems.

To allocate each job to only one end-system, the sum of each row in $ALLOCM$ (i.e., for each job) equal to 1. Then, the allocated end-systems are stored in the allocation array A and the neighbor switches of the end-systems are stored in SR.

$$\begin{aligned} &\forall j_1 \in \{1,...,j\}: \\ &\left(\sum_{r_1=1}^{ES} mat_{j_1,r_1}\right) = 1 \\ &\left(\bigvee_{r_1=1}^{ES} mat_{j_1,r_1} = 1 \rightarrow (a_{j_1} = r_1 \wedge sr_{j_1} = dr_{r_1})\right) \end{aligned} \tag{5.1}$$

To allow only one job to be allocated to an end-system, the sum for each end-system must be less than or equal to one.

$$\begin{aligned} &\forall r_1 \in \{1,...,ES\}: \\ &\left(\sum_{j_1=1}^{j} mat_{j_1,r_1}\right) \leq 1 \end{aligned} \tag{5.2}$$

The path of each message starts with the neighbor switch of the end-system that hosts the job, the first node for each message path $p_{1,1} \dots p_{m,1}$ is required to be the neighbor switch.

$$\begin{aligned} &\forall m_1 \in \{1,...,M\}, \forall j_1 \in \{1,...,J\}: \\ &s_{m_1} = j_1 \\ &\rightarrow \left(\bigvee_{r_1=1}^{ES} (a_{j_1} = r_1 \wedge p_{m_1,1} = dr_{r_1})\right) \end{aligned} \tag{5.3}$$

5.1.2.2 Connectivity Constraint

The path topology of the network is considered by these constraints based on the node connectivity C. Since an end-system is connected to only one switch, the connectivity constraints can be reduced by considering only the switches. If there is no direct connection between two switches a and b, then the path of a message must not include a hop from a to b.

$$\begin{aligned}
&\forall m_1 \in \{1,...,M\}, \forall r \in \{1,...,Max_H\}: \\
&h_{m_1} \geq r+1 \\
&\rightarrow \left(\bigvee_{a,b=ES+1}^{B} c_{a,b} = 1 \rightarrow \textit{Connected(a,b)} \right)
\end{aligned} \tag{5.4}$$

where the function *Connected()* states that a message's path is allowed to pass through the link between the two switches a and b.

$$\textit{Connected}(a,b) = (p_{m_1,r} = a \wedge p_{m_1,r+1} = b)$$

5.1.2.3 Collision Avoidance Constraint

These constraints are divided into three groups:

- Constraints to avoid collisions between a sending node and its neighbor switch
- Constraints to avoid collisions between switches
- Constraints to avoid collisions between a receiving node and its neighbor switch

The first constraints apply when a job sends more than one message. Since, there is only one link between any end-system and its neighbor switch, the constraints ensure that transmission times following the injection times I do not overlap.

$$\begin{aligned}
&\forall m_1 \in \{1,...,M\}, m_2 \in \{m_1+1,...,M\}, \\
&s_{m_1} = s_{m_2} \\
&\rightarrow (i_{m_1} \geq i_{m_2} + u_{m_2} \\
&\qquad \vee i_{m_2} \geq i_{m_1} + u_{m_1})
\end{aligned} \tag{5.5}$$

To prevent collisions of transmissions between switches, the scheduling of time-triggered messages ensures that no two messages are transmitted on one link at the same time. Thus, the messages should be transmitted on different paths or one needs to be scheduled before or after the transmission of the other message.

$$\begin{aligned}
&\forall m_1 \in \{1,...,M\}, m_2 \in \{m_1+1,...,M\},\\
&\ \forall r_1, r_2\ \in \{1,...,Max_H\}:\\
&\quad (p_{m_1,r_1} \neq p_{m_2,r_2} \vee p_{m_1,r_1+1} \neq p_{m_2,r_2+1}\\
&\quad \vee r_1+1 > h_{m_1} \vee r_2+1 > h_{m_2}\\
&\quad \vee i_{m_1} + (r_1+1)\cdot u_{m_1} \leq i_{m_2} + r_2 \cdot u_{m_2}\\
&\quad \vee i_{m_2} + (r_2+1)\cdot u_{m_2} \leq i_{m_1} + r_1 \cdot u_{m_1})
\end{aligned} \tag{5.6}$$

The third type of constraints is used when a job receives more than one message. Since there is only one link between an end-system and its neighbor switch, these constraints ensure that the transmission times of the messages from the neighbor switch to the end-system do not overlap.

$$\begin{aligned}
&\forall m_1 \in \{1,...,M\}, m_2 \in \{m_1+1,...,M\},\\
&\ j_1 \in \{1,...,J\}:\\
&\ d_{m_1,j_1} = 1 \wedge d_{m_2,j_1} = 1\\
&\ \rightarrow (i_{m_1} + (h_{m_1}+1)\cdot u_{m_1} \leq i_{m_2} + h_{m_2} * u_{m_2}\\
&\qquad \vee i_{m_2} + (h_{m_2}+1)\cdot u_{m_2} \leq i_{m_1} + h_{m_1} * u_{m_1})
\end{aligned} \tag{5.7}$$

5.1.3 Experimental Evaluation

Table 5.1 illustrates a comparison of the improved model against the previous model where the same scenarios conducted in section 4.3 are used. The table depicts the number of constraints as well as the execution time in seconds for both models for 22 different scenarios. The results show that the improved model has a significant reduction in constraints that leads to much less execution time. For example, scenario 12 consists of 2 switches and 6 end-systems where 4 jobs send 5 time-triggered messages. The number of constraints is reduced from 607 for the old model to 77 for the enhanced one and resulted in 0.25 seconds execution time compared to the previous model which needed 13.5 seconds. The last scenario shows a significant execution time reduction where the improved model finishes in 0.29 seconds whereas 150.98 seconds were needed by the previous model.

No.	Physical Model		Logical Model		Previous Model		Enhanced Model	
	Sw	Es	Job	Msgs	Constraints	Time (s)	Constraints	Time (s)
1	2	4	3	2	175	1.04	31	0.08
2	2	5	3	2	218	1.66	35	0.04
3	2	6	3	2	267	2.58	39	0.04
4	2	4	3	3	245	1.63	44	0.18
5	2	5	3	3	307	2.77	44	0.09
6	2	4	4	3	251	1.74	44	0.10
7	2	5	4	3	313	3.24	49	0.06
8	2	4	4	4	320	2.05	55	0.16
9	2	5	4	4	402	4.28	60	0.25
10	2	4	4	5	409	2.37	67	0.28
11	2	5	4	5	492	4.16	72	0.25
12	2	6	4	5	609	13.50	77	0.25
13	2	5	5	4	408	4.28	65	0.14
14	2	5	5	5	498	5.99	76	0.17
15	2	6	5	5	616	9.27	82	0.19
16	2	5	4	6	583	9.26	86	0.17
17	2	5	5	6	590	9.75	91	0.3
18	2	5	5	7	681	12.17	105	0.34
19	2	5	5	8	770	9.85	122	0.27
20	2	6	6	5	622	22.22	88	0.15
21	3	6	4	4	776	212.90	65	0.47
22	3	7	4	3	703	150.98	60	0.29

Table 5.1 Comparison of the enhanced model with the previous one.

5.2 Reliable Scheduling Model for Time-Triggered Messages

Hard real-time systems have one feature in common; a failure of a critical service can lead to deadly results, human life loss, high economic loss, or extensive environmental damage. Systems that incorporate this feature are called safety-critical systems. These systems require predictable, reliable, and real-time communication between the end-systems to ensure safety and reliability.

Reliability is defined as the likelihood of the failure-free operation of a system for a duration of mission in a specified environment [ALR+01]. Fault tolerant systems are based on the concept of FCRs where a system is partitioned into a set of subsystems, each of which will operate correctly regardless of any arbitrary fault outside the region [Kop11]. The goal of fault containment is to prevent the propagation of errors among system regions. A time-triggered network ensures the partitioning of the system into a set of independent FCRs, namely end-systems and time-triggered switches. One of the challenges is to tolerate faults introduced by communication links.

A parallel-series architecture can be used for reliability enhancement as depicted in Figure 5.2, where the links of a message path are shown in series with a certain number of redundant messages in parallel. In this paper, one redundant message is generated for every time-triggered message $(x = 2)$. If R_i is the reliability of the i-th link of the message path with $0 \leqslant R_i \leqslant 1$ then $(1 - R_i)^{x_i}$ is the failure rate if the i-th link is chosen with a redundancy degree expressed by x_i. Then, $1-(1-R_i)^{x_i}$ is the reliability of the i-th link and hence the message reliability R_s is [GLH+11]:

$$R_s = \prod_{i=1}^{n} \left[1-(1-R_i)^{x_i}\right] \tag{5.8}$$

Based on this formula, the use of the links over time can be optimized to maximize the reliability of each message. In order to find an optimum solution within a reasonable time, the above formula needs to be adopted to enable a linear optimization as follows [San15]:

$$\sum_{i=1}^{n} ln \left[1-(1-R_i)^{x_i}\right] \tag{5.9}$$

Moreover, the values in $ln\left[1-(1-R_i)^{x_i}\right]$ can be calculated before the scheduler is invoked. These pre-calculated values are given to the scheduler to simplify the scheduler functionality.

5.2.1 Scheduling Model For Fault-Tolerant Communication

This section illustrates a scheduling model that generates a schedule for a time-triggered network where each message is duplicated and sent to the destination through different hops. The implemented scheduling model is formulated using an MILP problem and solved by IBM CPLEX. This model comprises input constants for the network architecture along with the application in addition to the scheduling constraints.

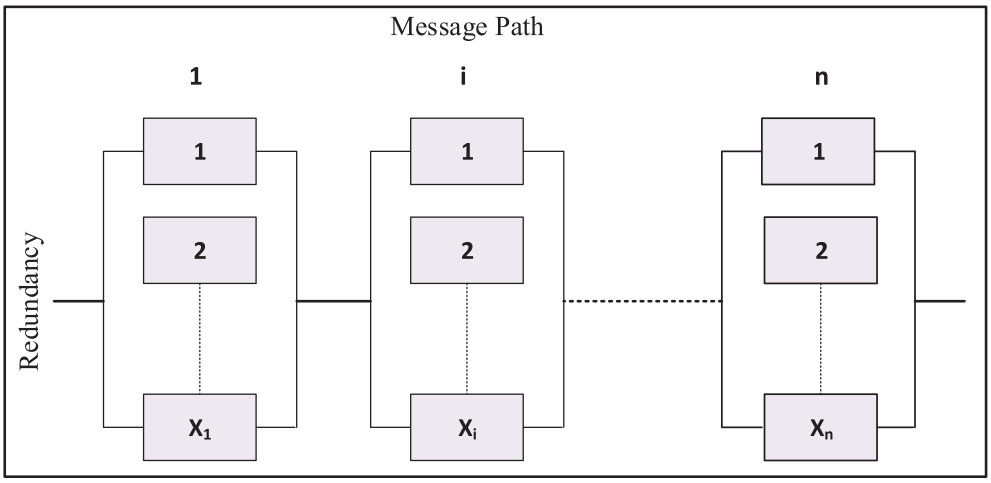


Fig. 5.2 Parallel-Series Redundancy Model

5.2.1.1 Constants

Table 5.2 shows the network architecture and the application model that are used as inputs in the formulation of the MILP problem. The physical model consists of the n nodes that are a set of s switches and e end-systems, thus $n = e + s$. These nodes are connected with bi-directional links in which a boolean two-dimensional array C of size $(n \cdot n)$ is used. This matrix is sorted where all end-systems come first before the switches. This helps to reduce the search space of the decision variables as well as the number of constraints, hence minimizing the computation time for scheduling [OM15]. Moreover, a link reliability matrix is introduced to capture the reliability of each link between two nodes. On the other hand, the logical model consists of the number of jobs denoted by j that send m messages where one end-system can be allocated to at most one job. The sender jobs can be represented as a one-dimensional array S. The receiving jobs are also described as a one-dimensional array D. The constant T denotes the period of the message according to the timing model of the time-triggered activities. The execution time of a job is denoted by E and assumed to be equal for all jobs.

5.2.1.2 Decision Variables

5.2.1.2.1 Neighbor switches To increase the reliability of the schedule, each end-system is connected to two switches where each message is injected into these neighbor switches. These neighbor switches can be expressed by a matrix with two rows for each allocated

Domain	Constant name	Description
Physical Model	n	Number of nodes
	C	Nodes Connectivity
	R	Links Reliability
Logical Model	j	Number of jobs
	m	Number of messages
	S	Sender jobs
	D	Destination jobs

Table 5.2 Overview table with constants for the CPLEX-based model

end-system *NS*.

$$NS = \begin{bmatrix} ns_{1,1} & \dots & ns_{j,1} \\ ns_{1,2} & \dots & ns_{j,2} \end{bmatrix} \in \{\mathbb{N}\}^{j\times 2}$$

5.2.1.2.2 Hop Count Each message is injected from the end-system that allocates a sender job to one of the neighbor switches along the intermediate switches and then to the allocated end-system of the destination job. The hop count H denotes the number of visited switches. In the absence of cyclic paths, the maximum hop count is $Max_H = s-1$.

$$H = \begin{bmatrix} h_1 \dots h_m \end{bmatrix}^T \in \{1,...,Max_H\}^m.$$

5.2.1.2.3 Injection Time The time by which the neighbor switch of the source job transmits a message is called the injection time which can be expressed using a one-dimensional array of size m.

$$I = \begin{bmatrix} i_1 \dots i_m \end{bmatrix}^T \in \{\mathbb{N}\}^m.$$

5.2.1.2.4 Path The path P denotes the switches that a message visits. A two-dimensional array is used where the rows denote the messages, and the columns represent the visited switches.

$$P = \begin{bmatrix} p_{1,1} & \cdots & p_{1,s} \\ \vdots & \ddots & \vdots \\ p_{m,1} & \cdots & p_{m,s} \end{bmatrix} \in \{1,...,n\}^{m\times s}$$

5.2.1.2.5 Link-Pair Reliability Every message, along with its redundant copy, traverses from the source node that allocates the sending job along a number of switches till it reaches the destination node. The scheduler keeps track of each link-pair a message and its redundant copy traverse and records its reliability in the decision variable REL.

$$REL = \begin{bmatrix} rel_{1,1} & \dots & rel_{1,Max_H} \\ \vdots & \ddots & \vdots \\ rel_{m,1} & \dots & rel_{m,Max_H} \end{bmatrix}^T \in \mathbb{R}^{m \times Max_H}$$

5.2.1.3 Scheduling Constraints

This part describes the constraints that are used in scheduling time-triggered messages.

5.2.1.3.1 Connectivity Constraints These constraints use the connectivity constants C in order to consider the network path topology. A message can traverse the link between two nodes a and b, only if there is a direct connection between them. These constraints are only executed for switches in order to reduce the number of constraints in the model.

$$\begin{aligned} &\forall m_1 \in \{1,...,m\}, \forall a \in \{s,...,n\}, \forall b \in \{s,...,n\}: \\ &c_{a,b} = 1 \\ &\rightarrow (h_{m_1} < 1) \vee \mathit{Traversed}(m_1, a, b), \end{aligned} \tag{5.10}$$

where m_1, a, and b are non-negative integers and *Traversed* states that a message m can visit the link between switches a and b.

5.2.1.3.2 Collision-Free Constraints These constraints ensure that only one message is processed by any node (end-system or switch). Each switch must receive only one message at a specific time from its directly connected switches. If more than one message must be sent from a certain switch and received by another switch, these messages need to be sent in disjoint time intervals.

$$\begin{aligned} &\forall m_1 \in \{1,...,m\}, m_2 \in \{m_1+1,...,m\}, \\ &\ \forall r_1, r_2 \in \{1,...,Max_H\}: \\ &\ \rightarrow (p_{m_1,r_1} \neq p_{m_2,r_2} \vee p_{m_1,r_1+1} \neq p_{m_2,r_2+1} \\ &\quad \vee i_{m_1} + (r_1+1) \cdot u_{m_1} < i_{m_2} + r_2 \cdot u_{m_2} \\ &\quad \vee i_{m_2} + (r_2+1) \cdot u_{m_2} < i_{m_1} + r_1 \cdot u_{m_1}) \end{aligned} \tag{5.11}$$

To prevent collisions between end-systems and their directly connected neighbor switches, different messages need to be injected/received at disjoint intervals.

5.2.1.3.3 Job Dependency Constraints When a message is sent to a job that needs to send another message, the latter message must be injected after the arrival of the former message. The start time of the message of the dependent job m_2 is after the receipt and execution for message m_1 of the relied upon job.

$$\begin{aligned}
&\forall m_1, m_2 \in \{1,...,m\}, \forall j_1 \in \{1,...,j\}: \\
&d_{m_1} = j_1 \wedge s_{m_2} = j_1 \\
&\rightarrow (i_{m_1} + h_{m_1} + e_{j_1} + 1 < i_{m_2}),
\end{aligned} \tag{5.12}$$

where m_1, m_2 denote the message numbers and j_1 denotes the job number as non-negative integers.

5.2.1.4 Reliability Constraints

The scheduling of the redundant time-triggered messages is done using pre-determined values of link-pair reliability *LPR*, where these values are calculated before CPLEX is invoked. If a pair of links is chosen for a message with its redundant copy, the reliability is determined for the specified hop.

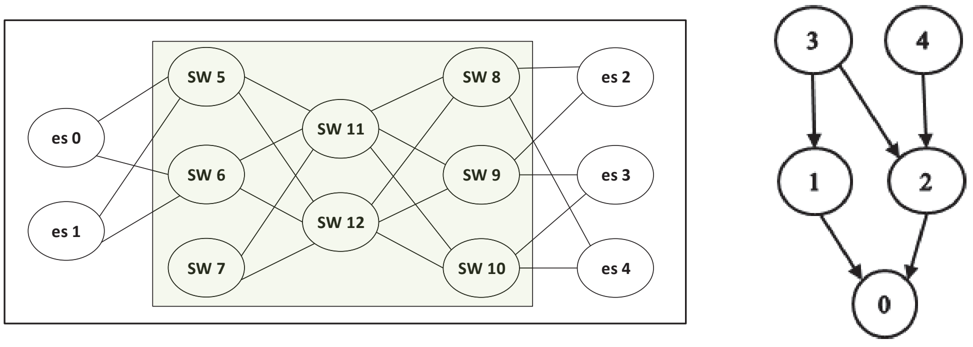
$$\begin{aligned}
&\forall m_1 \in \{1,...,m\}: \\
&\forall r_1 \in \{1,...,Max_H\}: \\
&\forall l_1, l_2 \in \{1,...,Links\}: \\
&l_1 \neq l_2 \wedge h_{m_1} > r_1 \wedge \mathit{check}(p_{m_1,r_1}, p_{mr_1,r_1}, l_1, l_2) \\
&\rightarrow rel_{m_1,r_1} = lpr(l_1, l_2)
\end{aligned} \tag{5.13}$$

The function $\mathit{check}(p_{m_1,r_1}, p_{mr_1,r_1}, l_1, l_2)$ tests whether the link-pair, l_1 and l_2, is traversed by a message m_1 and its redundant copy mr_1.

5.2.1.5 Transmission Delay Constraints

The typical objective in the scheduling of a time-triggered network is to generate a schedule where the makespan must be below the deadline. Therefore, the execution times of the jobs i_m and the message transmission delays h_m on the critical path, the message with the longest path, should be less than a real-time constraint N as shown in Equation (5.14)

$$\begin{aligned}
&\forall m_1 \in \{1,...,m\}: \\
&\quad CP[m_1] = (i_{m_1} + h_{m_1}) \\
&maximum(CP) < N.
\end{aligned} \tag{5.14}$$



(a) Physical Model (b) TT Logical Model

Fig. 5.3 Model example with 13 Nodes, 5 Jobs and 5 messages

5.2.1.6 Objective Function

The optimal scheduling for a reliable time-triggered network is to maximize the reliability of jobs. This is done by summing all the link-pair reliabilities for each message and its redundant copy along the whole path $Sum(rel_m)$ and maximizing it as shown in Equation (5.15).

$$\begin{aligned} \forall m_1 \ \in \{1,...,m\}: \\ CR[m_1] = Sum(rel_{m_1}) \\ maximize\Big(minimum(CR)\Big). \end{aligned} \tag{5.15}$$

5.2.2 Experimental Evaluation

This section evaluates the MILP model described in section 5.2.2. The experiments are based on scenarios established using SNAP library which is widely used in numerous academic researches [Les]. The scheduler takes the scenarios and provides a redundant schedule of the messages injected by jobs allocated to end-systems in such a way to maximize the reliability of the received messages. Since the scheduler is aimed to deal with time-triggered messages, the implemented constraints ensure the timing and satisfy the precedence requirements.

Consider a network with 13 nodes, 8 switches and 5 end-systems, where 5 jobs need to send 5 time-triggered messages. The corresponding physical and logical models are shown in Figure 5.3, where Figure a depicts the physical connection among nodes with bi-directional links and Figure b depicts the logical model. The nodes represent the jobs and the arrows represent the messages sent from one job to another one.

Table 5.3 summarizes the input constants for the physical and logical models illustrated in figure 5.3. The first row shows the number of end-systems, switches, jobs, and messages respectively. The second constant describes the link reliability of the node connectivity in the network with $0 \leqslant R_i \leqslant 1$. Most of the links have higher reliability except the links that connect nodes 8 and 11 as well as nodes 10 and 12 which have very low values 0.1. Constant 3 shows the jobs that send the messages, where each job can send more than one message. For example, job 1 is the sender of the first message and job 3 is the sender of the third and fourth messages. The last constant describes the receiving jobs of messages. For example, job 0 is the receiver of the first two messages.

Constant No.	Constant Name	Data
1	e, s, j, m	[5, 8, 5, 5]
2	R	[[0,0,0,0,0,0.9,0.9,0,0,0,0,0,0], [0,0,0,0,0,0,0.9,0.9,0,0,0,0,0], [0,0,0,0,0,0,0,0,0.9,0.9,0,0,0], [0,0,0,0,0,0,0,0,0,0.9,0.9,0,0], [0,0,0,0,0,0,0,0,0.9,0,0.9,0,0], [0.9,0,0,0,0,0,0,0,0,0,0,0.9,0.9], [0.9,0.9,0,0,0,0,0,0,0,0,0,0.9,0.9], [0,0.9,0,0,0,0,0,0,0,0,0,0.9,0.9], [0,0,0.9,0,0.9,0,0,0,0,0,0,0.1,0.9], [0,0,0.9,0.9,0,0,0,0,0,0,0,0.9,0.9], [0,0,0,0.9,0.9,0,0,0,0,0,0,0.9,0.1], [0,0,0,0,0,0.9,0.9,0.9,0.1,0.9,0.9,0,0], [0,0,0,0,0,0.9,0.9,0.9,0.9,0.9,0.1,0,0]]
3	S	[1,2,3,3,4]
4	D	[0,0,1,2,2]

Table 5.3 CPLEX Input constants for model in Figure 5.3

As illustrated in the previous section, the solution of a reliable scheduling problem in real-time distributed systems gives the best link-pairs for each message that maximizes the reliability. Since the number of link-pairs increases significantly with the number of links in the network, this makes the search space quite large for realistic network topologies which is illustrated in this section. The times of CPU calculation were obtained with CPLEX 12.6.1 running on a 12-core processor Intel(R) Xeon(R), 2.2 GHz server with the operating system Linux Ubuntu 14.04.1.

Table 5.4 depicts the results of the MILP model solved by CPLEX. It gave an optimal solution for the problem within 1400 seconds in which 1698 constraints are evaluated. Moreover, the output presented the values of the decision variables described in Section 5.2.1.2. The scheduler allocates end-systems for all jobs and generates a schedule for all messages as well as for the redundant copies. Here, job 1, which is allocated to end-system 3, sends one time-triggered message to job 0 which is allocated to end-system 2. This message is initiated at time $15\mu s$ in end-system 3 and passes through switches $9, 12$, and 8 and is then received by end-system 2 with a total of four hops between the sender and the receiver. Since link transmission times are assumed to be constant for all links $(3\mu s)$, the message is received at $15 + (4 \times 3) = 27\mu s$. Its redundant message is also initiated at time $15\mu s$ in end-system 3 and passes through switches $10, 11$, and 9 before end-system 2 receives it. Thus, every redundant message takes a different path between the sender and receiver compared to the first redundant message. Moreover, the scheduler did not choose the links between nodes [8, 11] and [10, 12], because they have a very low reliability.

To compare the reliable scheduling algorithm, the previous scenario was also solved using a random algorithm that does not take into account the link reliability. The right-hand side of Table 5.4 shows the results of this naive algorithm. Some messages $(2_r, 4$, and $4_r)$ use the low reliability links in their paths which reduces the probability of their successful arrival. This is clearly shown in the reliability field for message 4, where the reliability of the naive algorithm is 0.82 compared to the reliability of the proposed algorithm 0.98.

Since the scheduler deals with time-triggered messages, the scheduling of these messages is calculated in such a way that their transmissions are done without collisions. Moreover, there is a dependency between job 1 and job 3, where job 1 needs to send message 0 to job 0 while the former has to wait for job 3 to receive message 2. Thus, the transmission of message 0 in job 1, allocated to node 3, starts at the end of the transmission of message 2 in job 3 allocated to node 0. Thus, the injection time of message 0 should be after the reception time of message 2 and the execution time of this message in job 1 as shown in the following:

$$\begin{aligned} i_0 &= (i_2 + h_2 \cdot u_2 + e_1) \\ &= (0 + 4 \cdot 3 + 3) \\ &= 15\mu s \end{aligned}$$

Reliability algorithm							Random algorithm						
Job	Alloc. Node	Msg	Path	Start Time	Finish Time	Rel.	Job	Alloc. Node	Msg	Path	Start Time	Finish Time	Rel.
0	2						0	0					
1	3	0	3-9-12-8-2	15	27	0.98	1	4	0	4-8-12-5-0	15	27	0.98
		0_r	3-10-11-9-2	15	27				0_r	4-10-11-6-0	15	27	
2	1	1	1-6-12-8-2	18	30	0.98	2	2	1	2-8-12-5-0	18	30	0.98
		1_r	1-7-12-9-2	18	30				1_r	2-9-11-6-0	18	30	
3	0	2	0-5-12-9-3	0	12	0.98	3	1	2	1-6-12-8-4	0	12	0.90
		2_r	0-6-11-10-3	0	12				2_r	1-7-12-10-4	0	12	
		3	0-5-12-6-1	3	15	0.98			3	1-6-11-8-2	3	15	0.90
		3_r	0-6-12-7-1	0	12				3_r	1-7-12-9-2	3	15	
4	4	4	4-8-12-6-1	0	12	0.98	4	3	4	3-9-11-8-2	0	12	0.82
		4_r	4-10-11-7-1	0	12				4_r	3-10-12-9-2	0	12	

Table 5.4 Results for example model

In this chapter, extensions of the proposed model described in chapter 4 were given. First, optimizations to the previous model are presented in order to reduce the problem search space which minimizes the overall execution time of the scheduler as well as memory space. The experimental evaluation showed that the optimized model is more than 10 times faster than the previous model. This allows the scheduler to analyze example scenarios with a larger number of nodes.

After that, a reliable scheduling model for safety critical systems is presented in which messages are replicated and sent through different reliable paths. This model provides fault tolerance mechanisms in the presence of communication link failures and ensures reliable delivery of time-triggered messages. Moreover, the scheduling model obtains optimal schedules where the paths of the scheduled messages along with their redundant messages are guaranteed to have minimum end-to-end latency.

Chapter 6

Scheduling Model in Systems-of-Systems

This chapter provides an extended scheduling model explained in Chapter 4 that is used in the scheduling of real-time SoS applications. It starts with a conceptual model of the SoS architecture with its main two models namely, physical platform and logical application. Afterwords, the idea of incremental, distributed, and concurrent scheduling is discussed which gives an introduction to our proposed model [OM15]. Next, a formal description of SoS model in terms of physical as well as logical viewpoints. After that, a formal description of the allocation and scheduling functions in SoS is defined. Then, the SoS scheduling model is described using MILP. Finally, experimental evaluation for the proposed SoS model is discussed.

6.1 System-of-Systems Architecture

This section describes the SoS from logical and physical viewpoints. The introduced structural models are the basis for the subsequent formulation of the dynamic scheduling and allocation problem.

The overall conceptual model of the SoS is depicted in Figure 6.1. The SoS is comprised of constituent systems, where each constituent system is a distributed embedded systems, which is under the control of a given organization. Each constituent system consists of end-systems that are interconnected by real-time networks. Networks can include communication networks with different protocols and topologies (e.g., multi-star topology as depicted in Figure 6.1).

The interconnection of constituent systems occurs using a backbone communication infrastructures consisting of network domains. In analogy to the constituent systems, each network domain is within the responsibility of an organization that controls the resource allocations and their use by application subsystems. Technically, this control is realized by

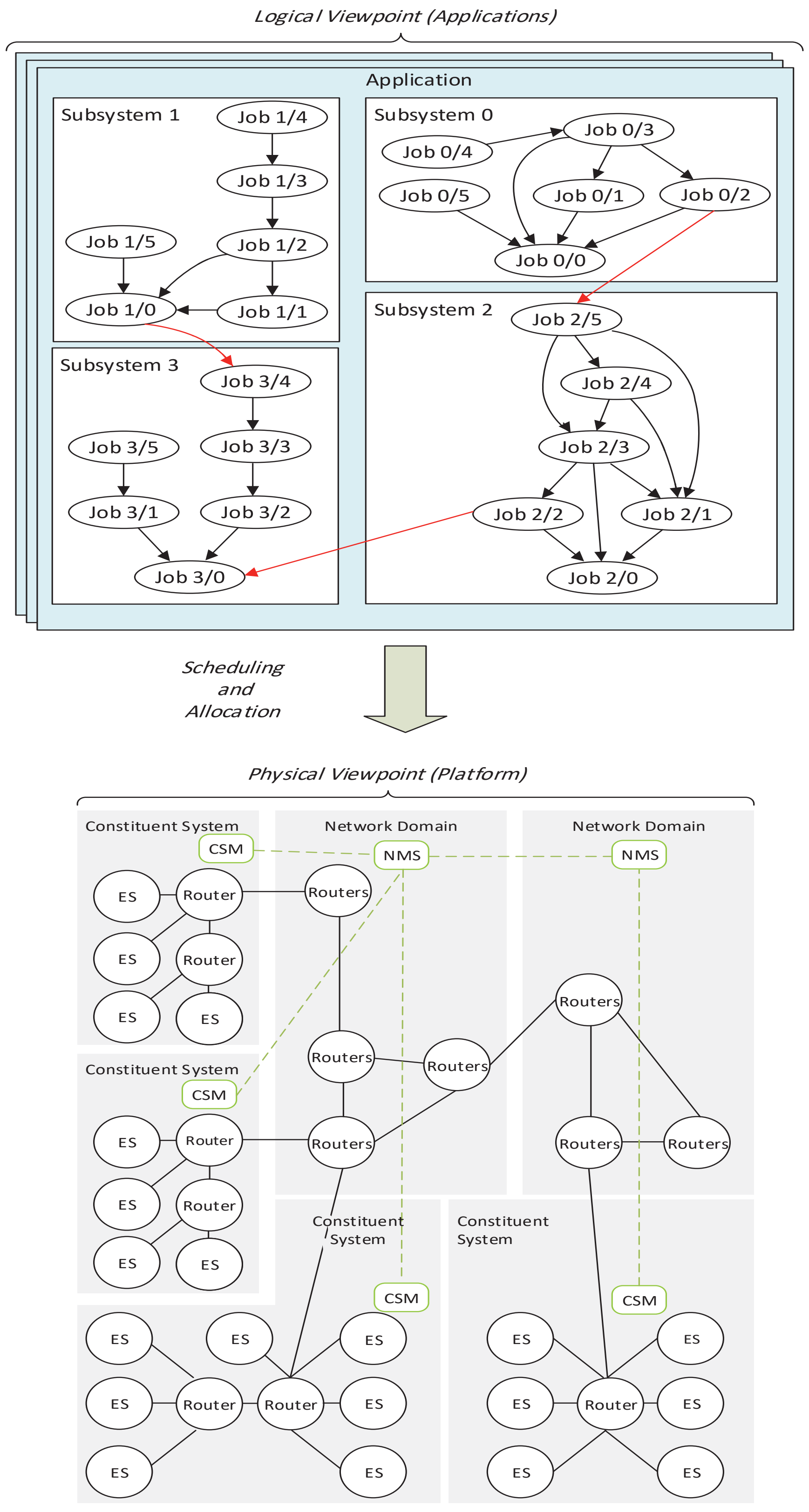


Fig. 6.1 Physical and Logical Viewpoint of the SoS

management services named Network Management Systems (NMS) of the network domains. The NMS configures the switches in the network domain, while also coordinating with other network domains and constituent systems.

Likewise, each constituent systems contains management services named CSM. The CSM performs the local configuration of the end-systems and networks within the constituent system. In addition, the CSM is responsible for the coordination with other constituent systems and network domains.

From a logical point of view, the SoS consists of applications, each of which is a hierarchical DAG with subsystems and services. The messages between subsystems and services represent the dependencies in the DAG. As an example, consider a medical application for health monitoring and patient care. This application involves different subsystems with respective services. A constituent system ’patient home’ hosts a subsystem ’health monitoring’ with local services (e.g., sensors, user interfaces). A constituent system ’hospital’ can provide a subsystem ’health alarm’ including local services for health records, the analysis of sensory data and the issuing of emergency treatment. A constituent system ’caregiver’ would offer a subsystem ’emergency response’ with services for remote interaction with patients.

From this example, we see the dynamic nature, large-scale, heterogeneity and lack of central control. Numerous of these medical applications will run in parallel for different patients, while sharing the infrastructure (e.g., network domains) and the constituent systems (e.g., hospitals, caregivers). In addition, other types of applications (e.g., energy management) will be active at the same time. The SoS is highly dynamic, e.g., when new patients are integrated into the system. The resource allocation also involves the coordination between different organizations (e.g., providers of network domain, hospitals, patients).

While the discovery and peering of services is addressed in previous work (e.g., service-oriented architectures [MTCM12], IoT-A [SCC+12], FIWARE IoT Discovery [FIW15]), the end-to-end resource allocation and scheduling for SoS involving real-time, reliability and safety requirements is an open research problem.

In this chapter, we provide a solution to this end-to-end resource allocation and scheduling based on the assumption of time-triggered protocols within the constituent systems and the network domains. This assumption is justified given the widespread use of time-triggered protocols in safety-relevant embedded systems (e.g., TTP in railway, TTEthernet in avionics, FlexRay in automotive) and the ongoing standardization activities for IEEE 802.1 [IEE15a], which introduces scheduled traffic based on time-triggered communication plans, while also offering run-time configurability and management capabilities. Likewise, TDMA with dynamic configuration capabilities is employed in protocols for the network domain (e.g., MPLS) for the resource allocation and quality-of-service guarantees.

The SoS is characterized by its dynamic nature, where applications are introduced at run-time. Therefore, communication resources and computational resources of the platform have to be dynamically allocated to the application. More precisely, the following decisions need to be taken for a new application:

- *SoS-level allocation:* Each application subsystem must be allocated to a constituent system.
- *SoS-level communication:* Messages between application subsystems must be mapped to paths between constituent systems along network domains.
- *Allocation within constituent systems:* Jobs must be allocated to end-systems within each constituent system.
- *Communication within constituent systems:* Messages between jobs of an application subsystem must be scheduled using paths between end-systems along switches.

In many safety-relevant systems, the inherent determinism of the time-triggered paradigm comes at the expenses of significantly reducing flexibility when adaptation to new events is required. For SoS, it is of crucial importance to dynamically adapt to the addition, change and removal of application services and physical building blocks (e.g., constituent systems, network domains). At the same time, we need to retain real-time and safety properties. Overcoming this limitation implies the ability of modifying the time-triggered schedule during runtime rather than precalculating offline schedules.

6.2 Incremental, Distributed, and Concurrent Scheduling

A naive approach relies on centrally computing new time-triggered schedules upon requests. However, the computation time needed to generate such a global schedule makes this approach unfeasible for fast-changing systems. In addition, SoS lack central information about the internal structure of all constituent systems.

Therefore, the following three principles are the foundation for the scheduling and allocation in SoSs:

- **Incremental scheduling.** In incremental scheduling, the transmission schedules of specific sending entities in the network are extended or modified whenever additional scheduled messages are required or whenever communication parameters are modified. An incremental transmission schedule thus does not completely replace an existing transmission schedule. However, it may modify some aspects of an existing transmission schedule to facilitate an incremental scheduling step. In order to achieve this, the incremental approach for deterministic networks should not require global knowledge about the overall network topology. The trade-off is that increasing the level of information about the network will result in better schedules at the expense of increased computation resources and network traffic for scheduling.

In this contribution, the incremental scheduling is driven by the dependencies imposed by the DAG of an application. An application subsystem can be scheduled after the relied upon subsystems have been scheduled. The dependencies comprise the messages between the application subsystem, where the transmission times determine the earliest possible start times for the dependent subsystems.

- **Distributed Scheduling.** Distributed scheduling reduces the overall scheduling time by parallelizing the search for a feasible solution using horizontal, vertical and diagonal partitioning schemes. We distribute the scheduling by computing the schedule of each application subsystem at the respective constituent system. The vertical partitioning of the scheduling problem results from the incremental scheduling steps of an application. In addition, the scheduling problem is horizontally partitioned along the different applications.

- **Concurrent Scheduling.** In a SoS many change requests can be requested and processed in parallel. Therefore, a SoS inherently requires concurrent scheduling of change requests while preserving the consistency in the configurations of constituent systems and network domains. For example, several new applications can be introduced at the same time as indicated in the medical monitoring scenario described above.

6.3 Problem Description

6.3.1 Platform Description

For the formal description of the physical viewpoint we introduce a set of end-systems ES, a set of constituent systems C, a set of network domains N and a set of switches SW. The elementary physical building blocks B (called nodes henceforth) are the switches and end-systems, whereas constituent systems and network domains are composite structures.

$$B = ES \cup SW \tag{6.1}$$

The platform is described by the following graph:

$$G_P =< V_P, E_P >, V_P = B, E_P = B \times B \tag{6.2}$$

Vertices are end-systems and switches, while edges represent the communication links between switches and constituent systems.

Each node either belongs a constituent system or it is part of a network domain of the SoS backbone infrastructure. This mapping is described by the following function f:

$$f_P : B \mapsto C \cup N \tag{6.3}$$

For a given constituent system or network domain, the nodes and the edges between these nodes must form a connected sub-graph of G_P.

Based on the constituent systems and network domains, we can define a high-level physical graph G_{HP} of the SoS.

$$G_{\text{HP}} =< V_{\text{HP}}, E_{\text{HP}} >, V_{\text{HP}} = C \cup N$$
$$E_{\text{HP}} = \{(e_1, e_2) | \exists \alpha, \beta \in E_P : f_P(\alpha) = e_1 \wedge f_P(\beta) = e_2\}$$

6.3.2 Application Description

From a logical point of view, the SoS consists of applications, where each application consists of jobs J that interact via the exchange of messages. An application A is described by the following DAG:

$$G_A =< V_A, E_A >, V_A = J, E_A \subseteq J \times J \tag{6.4}$$

The edges between the jobs are messages, which are exchanged between jobs.

Each application consists of application subsystems AS, which are connected sub-graphs of G_A. The mapping of jobs to application subsystems is described by the following function f_A:

$$f_A : J \mapsto AS \tag{6.5}$$

Based on the application subsystems, we can define a high-level logical graph G_{HA} of an application. This graph does not include jobs, but only application subsystems and the messages (i.e., edges) between application subsystems.

$$G_{\text{HA}} =< V_{\text{HA}}, E_{\text{HA}} >, V_{\text{HA}} = AS, E_{\text{HA}} \subseteq AS \times AS$$
$$E_{\text{HA}} = \{(e_1, e_2) | \exists \alpha, \beta \in E_A : f_A(\alpha) = e_1 \wedge f_A(\beta) = e_2\}$$

6.4 Formal Description of Scheduling and Allocation in SoS

Two levels of scheduling and allocation can be distinguished in SoSs. Firstly, application subsystems must be mapped to constituent systems. Secondly, the detailed scheduling and allocation of the jobs within each application subsystem can be performed.

6.4.1 High-Level Allocation of an Application

The first step of the allocation is the mapping of application subsystems to constituent systems:

$$\text{ALLOC}_{\text{AS}} : AS \mapsto C \tag{6.6}$$

Thereafter, each edge $< \alpha, \beta >$ (i.e., message) of the high-level application graph G_{HA} must be allocated to a path p in the high-level physical graph. Such a path in the high-level physical graph consists of a sequence of network domains from the constituent system of the sender α to the constituent system of the receiver β.

$$\begin{aligned} \text{ALLOC}_m : E_{\text{HA}} \mapsto p, E_{\text{HA}} =< \alpha, \beta >, p = (p_1, p_2, \dots, p_n) \\ p_1 = \text{ALLOC}_{\text{AS}}(\alpha) \\ p_n = \text{ALLOC}_{\text{AS}}(\beta) \\ \forall i \in \{1, 2, \dots, n-1\} :< p_i, p_{i+1} >\in E_{\text{HP}} \end{aligned}$$

6.4.2 Low-Level Allocation and Scheduling in Constituent Systems and Network Domains

For each application subsystem that is allocated to a constituent system c, the jobs $\bar{J}$ need to be allocated to the end-systems $\overline{ES}$ of c:

$$\begin{aligned} \text{ALLOC}_{\text{job,c}} : \bar{J} \mapsto \overline{ES} \\ \bar{J} = \{j \in J | f_A(j) = as \wedge \text{ALLOC}_{\text{AS}}(as) = c\} \\ \overline{ES} = \{es \in ES | f_P(es) = c\} \end{aligned} \tag{6.7}$$

Likewise, for each application subsystem that is allocated to a constituent system c the respective messages M must be mapped to paths and schedules:

$$\begin{aligned}
\text{SCHEDULE}_{m,c} : M \mapsto p, \\
M = \{<\alpha,\beta>\in E_A | f_A(\alpha) = f_A(\beta) = as \wedge \text{ALLOC}_{\text{AS}}(as) = c\} \\
p = (p_1, p_2, \ldots, p_n) \\
p_1 = \text{ALLOC}_{\text{job,c}}(\alpha) \\
p_n = \text{ALLOC}_{\text{job,c}}(\beta) \\
\forall i \in \{1,2,\ldots,n-1\} :< p_i, p_{i+1} >\in E_P, f_P(p_i) = f_P(p_k) = c
\end{aligned}$$

A message is an edge $<\alpha,\beta>$ in the DAG of the logical viewpoint. The respective jobs α and β must belong to the same application subsystem as that is allocated to a constituent system c. The links along the path $p_1, p_2, \ldots, p_n$ must be connected according to the graph G_P of the physical viewpoint.

6.5 Scheduling and Allocation Algorithm

The scheduling and allocation algorithm is summarized in Algorithm 1. The scheduling process is triggered by the arrival of a new application A. Initially, the allocation of subsystems to constituent systems $ALLOC_{\text{AS}}$ and the paths between constituent systems $ALLOC_m$ are determined. Thereafter, an enabled message is retrieved from the high-level application graph. A message is enabled if the relied upon application subsystems were already scheduled or if there are no relied upon application subsystems. In this case, the allocation and scheduling of the jobs and messages within the sending application subsystem as_1 is performed (i.e., $\text{ALLOC}_{\text{job,c}}$ and $\text{SCHEDULE}_{m,c}$). After the messages are scheduled on the network domains, the jobs and messages within the receiving application subsystem as_2 are scheduled.

6.6 Scheduling Model of SoS

This part presents the scheduling model for the incremental scheduling steps as introduced in the previous section. The model serves for the scheduling of an application subsystem in a constituent system according to Algorithm 1. Hence, the presented scheduling model serves for the local scheduling problem that needs to be solved by a CSM.

```
trigger : new application $A$ with $G_{\mathrm{HA}} =< V_{HA}, E_{HA} >$
determine ALLOC$_{\mathrm{AS}}$
determine ALLOC$_m$
$M_u$ = $E_{HA}$ // set of unscheduled messages
$M_a$ = $V_{HA}$ // set of unscheduled application subsystems
while $M_u \neq \varnothing$ do
    determine enabled messages $M_e \subseteq M_u$
    pick a message $m =< as_1, as_2 > \in M_e$
    // retrieve path
    $p \leftarrow$ ALLOC$_m(m)$
    // schedule sending application subsystem $as_1$
    // at constituent system $c = p_1$ (if unscheduled)
    if $as_1 \in M_a$ then
        incremental update of $ALLOC_{\mathrm{job,c}}$ for jobs in $as_1$
        incremental update of SCHED$_{\mathrm{m,c}}$ for msgs. in $as_1$
        $M_a \leftarrow M_a \setminus as_1$
    end
    // schedule network domains
    for $i \leftarrow 1$ to $n-1$ do
        incremental update of SCHED$_{\mathrm{m,n}}$ for <$p_i$, $p_{i+1}$ >
    end
    // schedule receiving application subsystem $as_2$
    // at constituent system $c = p_n$ (if unscheduled)
    if $as_2 \in M_a$ then
        incremental update of $ALLOC_{\mathrm{job,c}}$ for jobs in $as_2$
        incremental update of SCHED$_{\mathrm{m,c}}$ for msgs. in $as_2$
        $M_a \leftarrow M_a \setminus as_2$
    end
    $M_u \leftarrow M_u \setminus m$
end
```

Algorithm 1: Scheduling algorithm for new application A

The CSM needs to interact with other CSM as part of the distributed and incremental scheduling. In general, a subsystem will depend on messages from other subsystems and provide relied-upon messages to other subsystems. We denote these messages as border messages (red arrows in Figure 6.1) and we distinguish between incoming and outgoing border messages.

Figure 6.2 depicts an overview about the local scheduler in each CSM. In order for a CSM to schedule its allocated subsystem, it is required to have three main types of information. Namely, Logical and physical model of its located constituent system, reserved resources for previously scheduled application, and precedence constraints information about the dependent border messages from the other subsystems. After the CSM generates its local schedule, it updates its previous reserved resources database and notifies the other dependent constituent systems by sending timing information of their dependent border messages.

Table 6.1 depicts a summary of the constants with their associated domains. A typical constituent system based on switched Ethernet consists of a number of switches SW that can be interconnected in different topologies. Each of these switches has a number of end-systems that are connected in a star topology to the switch. The total number of end-systems is ES and the number of nodes of a constituent system is $B = ES + SW$. These nodes are interconnected using bi-directional physical communication links which can be described by a two-dimensional boolean array C, in which the B^2 values of the matrix are either 0 (not connected) or 1 (connected). In this work, the connectivity matrix is sorted where all end-systems come first and then the switches. This helps to reduce the computation time for scheduling.

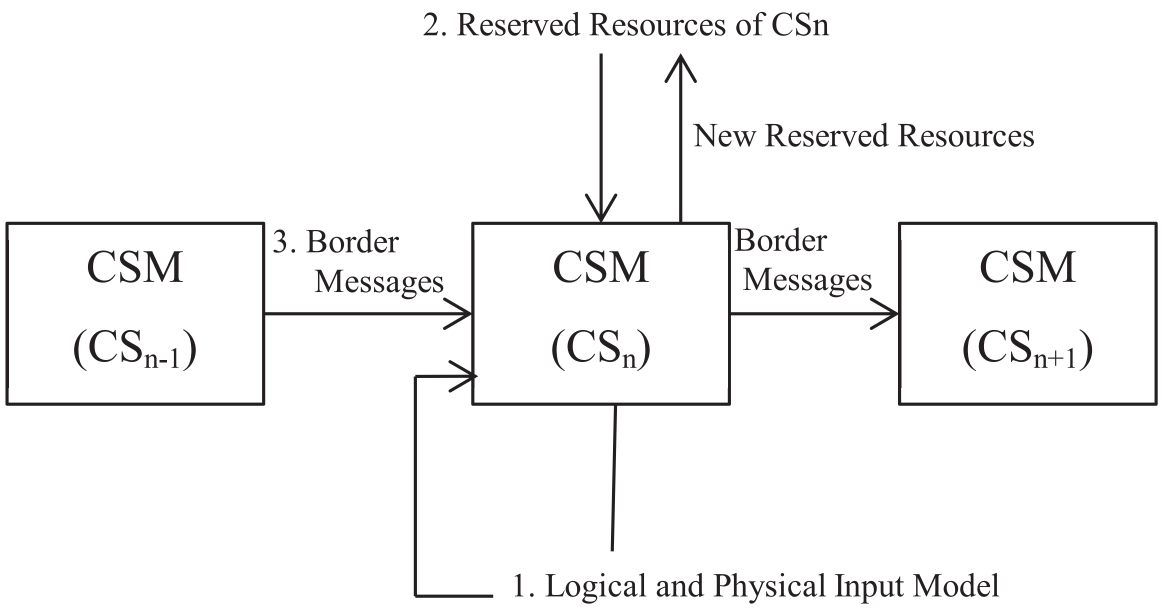


Fig. 6.2 Logical Viewpoint of Local Scheduler.

Domain	Constant name	Type	Description
	ES	$\mathbb{N}$	Number of end-systems
	SW	$\mathbb{N}$	Number of switches
	B	$ES+SW \in \mathbb{N}$	Numbers of nodes (ES+SW)
Constituent	BS	$\mathbb{N}$	Number of border switches
System	C	$\begin{bmatrix} c_{1,1} & \dots & c_{1,B} \\ \vdots & \ddots & \vdots \\ c_{B,1} & \dots & c_{B,B} \end{bmatrix} \in \{0,1\}^{N\times N}$	Node connectivity
	D_{Switch}	$[dr_1 \dots dr_{ES}]^T \in \{ES+1,...,B\}^{ES}$	Switch connected to an end-system
	U	$[u_1 \dots u_M]^T \in \mathbb{N}^M$	Hop transmission time
	J	$\mathbb{N}$	Number of jobs
	M	$\mathbb{N}$	Number of messages
	S	$[s_1 \dots s_M]^T \in \{1,...,J\}^M$	Sender jobs
Application	D	$\begin{bmatrix} d_{1,1} & \dots & d_{1,J} \\ \vdots & \ddots & \vdots \\ d_{M,1} & \dots & d_{M,J} \end{bmatrix} \in \{0,1\}^{M\times J}$	Destination jobs
	SN	$[sn_1,\dots,sn_M]^T \in \{0,1\}^M$	Vector denoting for each msg. whether of local origin
Subsystem	DN	$[dn_1,\dots,dn_M]^T \in \{0,1\}^M$	Vector denoting for each msg. whether with local destination
	INC	$\mathbb{N}$	Number of incoming border messages
	OUT	$\mathbb{N}$	Number of outgoing border messages
	E	$[e_1 \dots e_J]^T \in \mathbb{N}^J$	Job execution time
	GID	$[gid_1,\dots,gid_M]^T$	Global message ID

Table 6.1 Overview Input Table

6.6.1 Input

6.6.1.1 Input Model

To simulate the transmission and reception of border messages between applications in the proposed model, border switches BS are introduced in each constituent system. These switches are the access-points of constituent systems to the respective network domain. Conceptually, these switches allocate the jobs that either send or receive border messages. For better understanding and simplicity of the model, one border switch is introduced in each constituent system.

The connection of switches to end-systems is listed in a vector D_{Switch} that is determined by the connectivity matrix C. Each message requires a certain time, depending on the size of the message, to be transmitted on a link. Thus, every time a message is sent from one link to another one, its time is advanced by a hop transmission time U.

The application subsystem consists of a number of jobs J that communicate with each other by the exchange of M messages. These uni-directional messages are sent by the sending jobs, which are denoted by the vector S, where one job can send more than one message. These messages are received by jobs which can be specified in a two-dimensional boolean array D, where rows represent messages and columns represent receiving jobs. For example, $d_{2,4} = 1$ denotes that message 2 is sent to job 4.

When a message is transmitted inside a subsystem, the sender of this message is an end-system. On the other hand, when the message originates from outside the subsystem, then the sender of the message is modeled as a border switch in the scheduling problem. A boolean vector SN is used to specify whether a message is locally injected $(sn_m = 1)$ or from another constituent system $(sn_m = 0)$. Similarly, a boolean vector DN is used to differentiate between locally received messages $(dn_m = 1)$ or outgoing border messages $(dn_m = 0)$.

To keep track of the number of incoming border messages and outgoing border messages in each subsystem the constants INC and OUT are used respectively. Every message in the SoS has a unique identifier GID called the global message ID.

The computation time of jobs E is the execution time needed by the receiving job before sending a subsequent message.

6.6.1.2 Resource Information of CS

The introduction of a new application subsystem in a constituent system triggers dynamic reconfiguration by requiring a schedule for the additional jobs and messages. The CSM needs to calculate a new schedule for these jobs taking into account the reserved resources of previous schedules. Therefore, a multi-dimensional array Res is used to keep track of these reserved resources. The first and the second dimensions refer to the indices of the two nodes connecting the link (i.e., range $1 \dots B$). Finally, the third dimension denotes the index of the reservation of this link. Each link can have more than one reservation.

6.6.1.3 Scheduler State

Incremental scheduling in an SoS is the scheduling of messages that are transmitted between different subsystems. This requires information about transmission times of border messages in each subsystem in order to schedule these messages in the next subsystem.

A set of tuples BM is used that records the finish times of all border messages sent between subsystems. Each tuple contains two non-negative numbers, namely a global message ID and a finish time (ft). The finish time denotes the time by which an incoming border message is received at the border switch towards the other constituent systems.

$$BM = \{(gid_1, ft_1), (gid_2, ft_2), \dots\}$$
$$\text{with } gid_i \in \{1, 2, \dots, M\}, ft_i \in \mathbb{N}$$

6.6.2 Decision Variables

The local scheduler of the CSM generates two types of output information. A new schedule state for the new jobs and updated information about reserved resources. The latter is used to update the reserved resource database for subsequent scheduling steps.

6.6.2.1 New Schedule

This output information contains the schedule of the new jobs and messages. It consists of a schedule for time-triggered messages (i.e., mapping of jobs to end-systems, message paths) taking into account the dependencies with other local messages and border messages.

6.6.2.1.1 Job Allocation These variables denote the allocation of jobs to the nodes of the physical platform model. Jobs that send and receive local messages can only be allocated to end-systems while jobs that either send or receive border messages are allocated to the border switch. Since nodes are sorted with end-systems and border switches coming first, the maximum value a_i of an allocation variable is the sum of the numbers of end-systems and border switches.

$$A = [a_1 \dots a_J]^T \in \{1, ..., ES + BS\}^J$$

To ensure that each job is allocated to exactly one end-system, a boolean matrix $ALLOCM$ is used where the rows relate to jobs and columns to end-systems. For example, $mat_{3,2} = 1$ means that job 3 is allocated to end-system 2.

$$ALLOCM = \begin{bmatrix} mat_{1,1} & \dots & mat_{1,ES} \\ \vdots & \ddots & \vdots \\ mat_{J,1} & \dots & mat_{J,ES} \end{bmatrix} \in \{0,1\}^{J \times ES}$$

To keep track of switches via which a job can transmit a message we use a vector SW. The vector SW denotes for each job an access-point switch that is directly accessible from the end-system where the job is located. All other switches can only be reached by more than one hop. For example, $sr_2 = 14$ denotes that the switch with ID 14 is the access-point switch for the end-system hosting the job 2.

$$SR = [sr_1 \dots sr_J]^T \in \{Z, ..., B\}^J$$

where $Z = ES + BS + 1$.

6.6.2.1.2 Hop Count A message is injected at the source end-system, where the sender job was allocated. It is then transported along one or more switches before being received by the end-system of the destination job. In order to express the number of visited switches for each message after the access-point switch the vector hop count H is used and the maximum value of its elements denotes the critical path length. In the absence of cyclic paths, the maximum path length is $\max_H = SW - 1$.

$$H = [h_1 \dots h_M]^T \in \{1, ..., \max_H\}^M$$

6.6.2.1.3 Injection Time This one-dimensional array represents the times by which the messages are injected in the network of the constituent system. To reduce the search space of the model and since there is only one path between any node and its neighbor switch, this variable records the transmission time of a message starting from the neighbor switch of the sender's end-system where it is rescaled when a schedule is generated.

$$I = [i_1 \dots i_M]^T \in \{1, ..., \mathbb{N}\}^M$$

6.6.2.1.4 Path and Visited Switches To record the path between the message's source and destination end-system, the path array P is used. Since the sending and the receiving jobs are known beforehand, each row represents the path of a message starting from the switch connected to the end-system which allocates a source job to the switch connected to the end-system in which the destination job is allocated. For example, $p_{1,2} = 14$ means that the second switch that message number 1 visits is the node with ID 14. The maximum number of nodes in a path equals the maximum number of hops.

$$P = \begin{bmatrix} p_{1,1} & \cdots & p_{1,SW} \\ \vdots & \ddots & \vdots \\ p_{M,1} & \cdots & p_{M,SW} \end{bmatrix} \in \{Z,...,B\}^{M\times SW}$$

where $Z = ES + 1$.

For the purpose of calculating the end-to-end latency, a boolean matrix O is used to denote the switches that are passed by a message. For example, $o_{2,3} = 1$ means that message 2 travels through a switch with ID 3.

$$O = \begin{bmatrix} o_{1,1} & \cdots & o_{1,SW} \\ \vdots & \ddots & \vdots \\ o_{M,1} & \cdots & o_{M\times SW} \end{bmatrix} \in \{0,1\}^{M\times SW}$$

6.6.2.2 Reserved Resources

After a schedule is generated, the transmission links for paths of all messages are used to update a reserved resources database Res. Each entry in this database consists of the IDs of the end-systems and/or switches connecting the reserved link in addition to the start time of a message at the specified link. For example, $Res_{3,5,2} = 10$ denotes that the link connecting nodes 3 and 5 has two reservations. The second reservation starts at 10 ms and has a duration of the transmission time of the message u_m.

6.6.3 Scheduling Constraints

This part describes the constraints that are used in the scheduling of time-triggered messages in a constituent system.

6.6.3.1 Distributed Scheduling Constraints

As a prerequisite for the distributed scheduling, the CSM requires information about the transmission times of border messages that are exchanged between different application subsystems.

The injection times of local messages ($sn_m = 1$) as well as incoming border messages ($sn_m = 0$) in a subsystem can be evaluated as follows:

$$\begin{aligned}
&\forall m_i \in \{1,...,M\}: \\
&\quad sn_{m_i} = 1 \rightarrow i_{m_i} \geq u_{m_i} \\
&\quad sn_{m_i} = 0 \rightarrow i_{m_i} \geq ft_{m_i} \text{ where } (gid_{m_i}, ft_{m_i}) \in BM
\end{aligned} \tag{6.8}$$

6.6.3.2 Incremental Scheduling Constraints

New applications introduce additional jobs where the new schedule must take into account the reserved resources of the previous schedule as denoted by Res. These corresponding constraints can be divided into three groups:

- Constraints for links between sending end-systems and their access-point switches
- Constraints for links among switches
- Constraints for links between receiving end-systems and their access-point switches

6.6.3.2.1 Reserved resources between sending end-systems and their access-point switches End-systems are connected to switches in a star topology. Hence, if the sender is an end-system, it means that there is only one link where the first node is an end-system and the second node is its access-point switch.

$$\begin{aligned}
&\forall m_1 \in \{1,...,M\}, \forall r_1, r_2 \in \{1,...,B\}: \\
&\quad (Res_{r_1,r_2,0} \geq 0) \wedge (r_1 \leq ES) \wedge (sn_{m_1} = 1) \\
&\quad \rightarrow \Bigg((a_{s_{m_1}} \neq r_1 \vee p_{m_1,0} \neq r_2) \\
&\qquad\quad \vee \Big(\bigwedge_{z=1}^{M} (i_{m_1} + u_{m_1} \leq Res_{r_1,r_2,z}) \\
&\quad \vee (i_{m_1} - u_{m_1} \geq Res_{r_1,r_2,z}) \Big) \Bigg)
\end{aligned} \tag{6.9}$$

6.6.3.2.2 Reserved resources among switches Since the connections of the switches can have different topologies, all possible paths need to be checked regarding the reserved resources.

$$
\begin{aligned}
&\forall m_1 \in \{1,...,M\}, \forall r_1, r_2 \in \{1,...,B\}: \\
&(Res_{r_1,r_2,0} \geq 0) \wedge (r_1 > ES) \wedge (r_2 > ES) \\
&\rightarrow \left(\bigvee_{r_3=2}^{R} \Big((p_{m_1,r_3-1} \neq r_1 \vee p_{m_1,r_3} \neq r_2) \right. \\
&\wedge (p_{m_1,r_3-1} \neq r_2 \vee p_{m_1,r_3} \neq r_1) \Big) \\
&\qquad \vee (h_{m_1 < r_3}) \\
&\qquad \vee \Big(\bigwedge_{z=1}^{M} (i_{m_1} + r_3 \cdot u_{m_1} \leq Res_{r_1,r_2,z}) \\
&\left. \vee (i_{m_1} + (r_3 - 1) \cdot u_{m_1} \geq Res_{r_1,r_2,z} + u_{m_1}) \Big) \right)
\end{aligned}
\tag{6.10}
$$

6.6.3.2.3 Reserved resources between receiving end-systems and their access-point switches Again, if the receiving node is an end-system, it means that there is only one link where the first node is an end-system and the second node is its access-point switch.

$$
\begin{aligned}
&\forall m_1 \in \{1,...,M\}, \forall r_1, r_2 \in \{1,...,B\}: \\
&(Res_{r_1,r_2,0} \geq 0) \\
&\rightarrow \forall j_1 \in \{1,...,J\}: \\
&\quad d_{m_1,j_1} = 1 \\
&\rightarrow \left((a_{j_1} \neq r_1 \vee p_{m_1,0} \neq r_2) \right. \\
&\qquad \vee \Big(\bigwedge_{z=1}^{M} (i_{m_1} + r_3 \cdot u_{m_1} \leq Res_{r_1,r_2,z}) \\
&\left. \vee (i_{m_1} + (r_3 - 1) \cdot u_{m_1} \geq Res_{r_1,r_2,z} + u_{m_1}) \Big) \right)
\end{aligned}
\tag{6.11}
$$

6.6.3.3 Connectivity Constraint

The first constraint considers the path topology of the network based on the node connectivity C. Since an end-system is connected to only one switch, the connectivity constraints can be reduced by considering only the switches. If there is no direct connection between two switches a and b, then the path of a message must not include a hop from a to b.

$$\begin{aligned}
&\forall m_1 \in \{1,...,M\}, \forall r \in \{1,...,Max_H\}: \\
&h_{m_1} \geq r+1 \\
&\rightarrow \left(\bigvee_{a,b=ES+1}^{B} c_{a,b} = 1 \rightarrow \textit{Connected(a,b)} \right)
\end{aligned} \tag{6.12}$$

where the function *Connected()* states that a message's path is allowed to pass through the link between the two switches a and b.

$$\textit{Connected}(a,b) = (p_{m_1,r} = a \wedge p_{m_1,r+1} = b)$$

6.6.3.4 Collision Avoidance Constraint

These constraints are divided into three groups:

- Constraints to avoid collisions between a sending node and its access-point switch
- Constraints to avoid collisions between switches
- Constraints to avoid collisions between a receiving node and its access-point switch

The first constraints apply when a job sends more than one message. Since there is only one link between any end-system and its access-point switch, the constraints ensure that transmission times following the injection times I do not overlap.

$$\begin{aligned}
&\forall m_1 \in \{1,...,M\}, m_2 \in \{m_1+1,...,M\}, \\
&s_{m_1} = s_{m_2} \\
&\rightarrow (i_{m_1} \geq i_{m_2} + u_{m_2} \\
&\qquad \vee i_{m_2} \geq i_{m_1} + u_{m_1})
\end{aligned} \tag{6.13}$$

To prevent collisions of transmissions between switches, the scheduling of time-triggered messages ensures that no two messages are transmitted on one link at the same time. Thus, the messages should be transmitted on different paths or one needs to be scheduled before or after the transmission of the other message.

$$\begin{aligned}
&\forall m_1 \in \{1,...,M\}, m_2 \in \{m_1+1,...,M\},\\
&\quad \forall r_1, r_2 \in \{1,...,Max_H\}:\\
&\qquad (p_{m_1,r_1} \neq p_{m_2,r_2} \vee p_{m_1,r_1+1} \neq p_{m_2,r_2+1}\\
&\qquad \vee r_1+1 > h_{m_1} \vee r_2+1 > h_{m_2}\\
&\qquad \vee i_{m_1} + (r_1+1)\cdot u_{m_1} \leq i_{m_2} + r_2 \cdot u_{m_2}\\
&\qquad \vee i_{m_2} + (r_2+1)\cdot u_{m_2} \leq i_{m_1} + r_1 \cdot u_{m_1})
\end{aligned} \tag{6.14}$$

The third type of constraints is used when a job receives more than one message. Since there is only one link between an end-system and its access-point switch, these constraints ensure that the transmission times of the messages from the access-point switch to the end-system do not overlap.

$$\begin{aligned}
&\forall m_1 \in \{1,...,M\}, m_2 \in \{m_1+1,...,M\},\\
&\quad j_1 \in \{1,...,J\}:\\
&\quad d_{m_1,j_1} = 1 \wedge d_{m_2,j_1} = 1\\
&\quad \rightarrow (i_{m_1} + (h_{m_1}+1)\cdot u_{m_1} \leq i_{m_2} + h_{m_2} * u_{m_2}\\
&\qquad \vee i_{m_2} + (h_{m_2}+1)\cdot u_{m_2} \leq i_{m_1} + h_{m_1} * u_{m_1})
\end{aligned} \tag{6.15}$$

6.6.3.5 Job Dependency Constraint

Depending on the precedence constraints between the jobs, jobs may need to wait for the output of the transmission of other jobs before they begin the transmission. This constraint ensures that if a job sends a message m_1 to another job that needs the output of m_1 in order to send m_2, the start time of m_2 must be after the end of the transmission and execution of m_1.

$$\begin{aligned}
&\forall m_1, m_2 \in \{1,...,m\}, \forall j_1 \in \{1,...,j\}:\\
&\quad d_{m_1,j_1} = 1 \ \wedge \ s_{m_2} = j_1\\
&\quad \rightarrow i_{m_1} + (h_{m_1}+1)\cdot u_{m_1} + e_{j_1} < i_{m_2}
\end{aligned} \tag{6.16}$$

Each message must reach the destination node within its path and the selected number of hops.

$$\begin{aligned}
&\forall m_1 \in \{1,...,M\}, \forall j_1 \in \{1,...,J\}\\
&d_{m_1,j_1} = 1\\
&\quad \rightarrow \left(\bigvee_{r_1=1}^{W} (p_{m_1,r_1} = sr_{j_1} \ \wedge \ r_1 = h_{m_1}) \right)
\end{aligned} \tag{6.17}$$

6.6.3.6 Job Assignment Constraints

These constraints ensure that a job can be assigned to only one end-system. This is done by having the sum of each row in $ALLOCM$ (i.e., for each job) equal to 1. Then, the allocated end-systems are stored in the allocation array A and the access-point switches of the end-systems are stored in SR.

$$\begin{aligned}
&\forall j_1 \in \{1,...,j\}: \\
&\left(\sum_{r_1=1}^{ES} mat_{j_1,r_1} \right) = 1 \\
&\left(\bigvee_{r_1=1}^{ES} mat_{j_1,r_1} = 1 \rightarrow (a_{j_1} = r_1 \wedge sr_{j_1} = dr_{r_1}) \right)
\end{aligned} \tag{6.18}$$

To allow only one job to be allocated to an end-system, the sum for each end-system must be less than or equal to one.

$$\begin{aligned}
&\forall r_1 \in \{1,...,ES\}: \\
&\left(\sum_{j_1=1}^{j} mat_{j_1,r_1} \right) \leq 1
\end{aligned} \tag{6.19}$$

In order to start the path of each message with the access-point switch of the end-system that hosts the job, the first node for each message path $p_{1,1} \ldots p_{m,1}$ is required to be the access-point switch.

$$\begin{aligned}
&\forall m_1 \in \{1,...,M\}, \forall j_1 \in \{1,...,J\}: \\
&s_{m_1} = j_1 \\
&\rightarrow \left(\bigvee_{r_1=1}^{ES} (a_{j_1} = r_1 \wedge p_{m_1,1} = dr_{r_1} \right)
\end{aligned} \tag{6.20}$$

6.6.4 Objective Function

The objective is to minimize the maximum transmission time of the time-triggered messages (i.e., minimization of critical path). This is done by first finding the transmission time of each time-triggered message, expressed as the sum of the injection time i_m and the number of hops h_m multiplied by the transmission duration of a message u_m. Then, the objective function minimizes the highest value among all these messages.

$$
\begin{aligned}
&\forall m_1 \in \{1, ..., M\} : \\
&\qquad CP[m_1] = (i_{m_1} + h_{m_1} \cdot u_{m_1}) \\
&minimize \quad max(CP)
\end{aligned}
\tag{6.21}
$$

6.7 Experimental Evaluation

This section discusses the experimental evaluation. A generator for example scenarios and a high-level scheduler were implemented to validate the scheduling problem.

6.7.1 Generator for Example Scenarios

A generic SoS generator was realized to build example scenarios for the evaluation of the proposed scheduling models. Based on input parameters, the generator creates random platforms and applications according to the conceptual model introduced in Section 6.1. The input parameters for the physical viewpoint include the desired number of constituent systems and network domains, the average number of end systems and switches per constituent system, and the average node degree of the switches. In the logical viewpoint, input parameters are the desired number of applications, the average number of subsystems per application, the number of jobs per application subsystem and the average node degree of jobs.

The SNAP [Les] library was used for the generation of DAGs and undirected graphs in the generations. A DAG is required for the graph of jobs in each application subsystem as well as for the interconnection of application subsystems. The undirected graphs describe the connectivity of the switches in constituent systems as well as the interconnection of network domains and constituent systems. The outputs are visualized using the GraphViz library.

An example of a generated scenario is shown in figure 6.3. The figure depicts in detail one of the constituent systems with 20 end-systems and switches where node ID 15 is a border switch; it has also one of the application subsystems with 6 jobs. The jobs of the application subsystem 1 send 6 local messages and one border message. In addition, one incoming border message is received from another constituent system.

6.7.2 High-Level Scheduler

A high-Level scheduler was implemented to evaluate the conceptual model and the scheduling problem. This high-level scheduler implements Algorithm 1 by performing a random

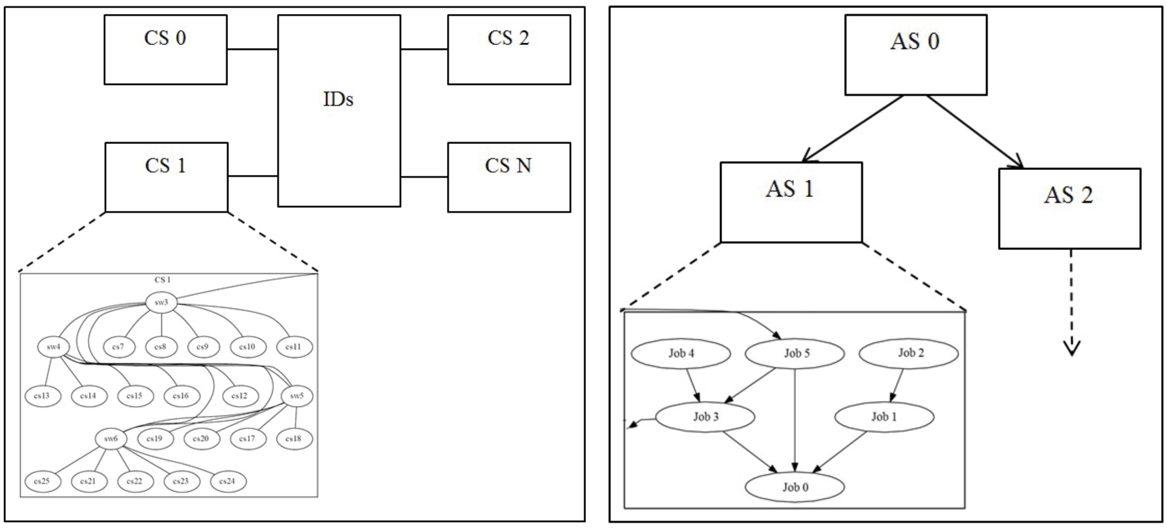


Fig. 6.3 Generated Platform and Application

allocation of application subsystems to constituent systems. The paths between application subsystems are determined by computing the shortest paths. An extension of the high-level scheduler with support for an optimized allocation of application subsystems and path determination at the SoS-level is planned as future work.

The output of the high-level scheduler are CPLEX scheduling models with the constants, constraints and decision variables as introduced in Section 6.5.

6.7.3 Results

Table 6.2 depicts the scheduling time for three different SoSs. Every SoS consists of seven applications each containing four application subsystems. The computation times were obtained with CPLEX 12.6.1 running on a 12 processor Intel(R) Xeon(R), 2.2 GHz server with the operating system Linux Ubuntu 14.04.1. CPLEX was used for the local scheduling in each constituent system, either stopping after a feasible solution is found or computing an optimal local schedule.

The scheduling time is measured in seconds to find a feasible solution as well as an optimal solution. The finish times in the table denote the makespans of the respective applications in ms. As can be seen in the table, in some cases the optimal local schedule leads to increased makespans of later applications. The reason is the unavailability of early time slots for messages of subsequent applications.

The results have demonstrated that hard real-time constraints can be satisfied using time-triggered messages in SoS. This is done using incremental, distributed, and concurrent scheduling.

Scenario	Logical Viewpoint			Physical Viewpoint			Finish Time		Execution Time	
	Applications	Jobs	Messages	CS	Endsystems	Switches	Feasible	Optimal	Feasible	Optimal
	1	30	30				39	36	3.08	3.48
	2	30	38				51	48	11.96	27.76
	3	30	30				69	42	14.42	13.55
SoS 1	4	30	26	4	75	21	63	42	19.47	95.31
	5	30	30				87	84	112.5	331.23
	6	30	30				114	102	136.95	522.36
	7	30	30				96	90	236.21	737.3
	1	30	30				39	33	3.72	6.41
	2	30	30				48	60	10.69	11.86
	3	30	30				45	36	10.34	17.91
SoS 2	4	30	30	4	85	23	123	117	49.61	50.41
	5	30	26				60	54	81.75	98.74
	6	30	26				90	69	39.18	41.39
	7	30	26				75	81	298.76	545
	1	30	26				54	51	2.59	2.93
	2	30	30				72	72	4.52	6.45
	3	30	38				78	60	33.56	105.5
SoS 3	4	30	30	4	74	21	66	66	32.34	520.3
	5	30	26				90	84	21.25	111.33
	6	30	26				72	81	108.37	452.19
	7	30	26				81	93	351.8	692

Table 6.2 Results of Different SoS Scenarios

The incremental scheduling allows the scheduler to reserve the resources from previous applications such as end-system processing intervals and communication link transmission times in order to avoid contention with the newly generated application schedules. The distributed cooperation among the different CSMs is a prerequisite for supporting the autonomy and lack of central control in SoS on one hand and simplifying the scheduling problem on the other hand by executing each application subsystem in its allocated constituent system which reduces the overall execution time and memory usage.

Chapter 7

Validation Framework for Time-Triggered Systems-of-Systems

Compared to monolithic systems, SoSs are based on a number of operationally and administratively independent, evolutionary developed, and graphically distributed constituent systems [Mai98]. Thus, it is difficult to set up such systems and a simulation environment approach is needed for verifying such complex systems. This chapter describes a simulation framework of SoSs where time-triggered application schedules can be verified and analyzed. First, it gives an introduction for simulation framework to be described. Then, the models of the SoS models are explained, namely the generic building blocks for TTEthernet system elements, the configuration file of nodes, and the CSM unit. After that, a description of the tool chain that is used to generate understandable information schedule for every application request, configures the related constituent system nodes, and analyzes provided results [MAO17]. Next, the integration of the SoS scheduling model in the CSM is presented in which the scheduled applications are fed to the simulation framework during its execution. This simulates the execution of SoS applications and enhances the analysis of the incremental schedule. Finally, experimental scenarios are conducted and their results are evaluated.

7.1 SoS Simulation Framework

A naive approach for scheduling SoS applications relies on centrally computing new time-triggered schedules upon requests. However, the computation time needed to generate such a global schedule makes this approach unfeasible for fast-changing systems. In addition, SoS lack central information about the internal structure of all constituent systems.

Real-time support in SoSs is an open research challenge due to the lack of central control as well as the evolving and dynamic nature of the interactions between the constituent systems. Real-time support is essential in many safety-relevant application areas such as smart city, medical, military and industrial SoS. This SoS architecture is realized using a TTEthernet model building blocks (e.g., TTEthernet switches and TTEthernet end-systems) that are used to provide real-time requirements and mixed criticality applications [AO13].

This simulation environment uses the OPNET tool suite for discrete event simulations of TTEthernet communication networks [OPN]. Simulation models in OPNET are organized hierarchically consisting of four main levels: the SoS network, constituent systems, node models and process models.

The top level refers to the SoS network which contains a number of constituent system models and a network domain, that connects these constituent systems with each other, using building blocks from the standard library and user-defined components. At this level, statistics about the network are collected, the simulation is executed and results are viewed. The second level is the constituent system that is implemented using the time-triggered nodes (i.e. end-systems and switches). The node models are at the third level in the hierarchy and have a modular structure. The node is defined by connecting various modules with packet streams. The connections between modules allow packets and status information to be exchanged between modules. The modules in the nodes are implemented by using process models, the lowest level in the hierarchy. Process models are represented by finite state machines, and a process interface that defines the parameters for interfacing other process models and configuration attributes. Finite state machine models are described as embedded C or C++ code blocks. The hierarchical structure of the models, coupled with support for C and C++ code, allows for easy development of communication or networking models.

7.1.1 SoS models

7.1.1.1 TTEthernet Nodes

The main simulation building blocks are generic building blocks of the infrastructure elements of a TTEthernet-system, which can be configured and extended to create an application-specific simulation model:

- Generic model of a TTEthernet switch. TTEthernet switches are central building blocks of a TTEthernet-based system. A generic simulation model of a TTEthernet switch supporting time-triggered, rate-constraint, and best effort communication is developed. In order to construct the overall simulation model, the user can perform multiple instantiations of the generic switch, establish connections to end-systems

Switch Configuration File

Traffic Type	VL-ID	Period	Phase/	Sender Port	Receiver Port	Size

End-systems Configuration File

Node ID	VL-ID	Period	Phase/ Jitter	Size

Table 7.1 OPNET Configuration Files

and other switches, and assign to each switch instantiation a corresponding configuration. The switch configuration defines the message timing including a time-triggered communication plan.

- Generic model of a TTEthernet end-system. TTEthernet end-system are the communication end points within the TTEthernet system. The user can perform instantiates of the generic TTEthernet end-system and connect each instantiation to TTEthernet switches. End-system can be configured to produce messages according to application-specific parameters (e.g., interarrival time, distributions of rate constraint messages, periods of time-triggered messages). In addition, nodes can be extended with the application behavior (e.g., C++ application code).

7.1.1.2 Configuration Files

Table 7.1 illustrates the fields of a configuration file for each switch and for end-systems. In a switch configuration, a traffic type, either time-triggered or rate-constrained, is needed to determine the traffic policy of the message. Then, a virtual link ID and a period need to be assigned for each message. After that, the phase time of time-triggered messages or minimum inter-arrival time for rate-constrained messages is needed when the message arrives to the switch. At this time, detailed information is required about the switch's sending and receiving port for this message. Finally, the size of each message needs to be specified.

On the other hand, the node configuration file represents the reserved resources for all end-systems in the SoS. Each row represents a message transmission or reception for an end-system. The first field is an end-system's ID and then the virtual link ID of the message. Then, the period of the message and the time by which a message is injected/received are specified. The fifth parameter is the message size.

7.1.1.3 Constituent System Manager

This unit is located in each constituent system and is responsible for assigning the configuration parameters to the end-systems and switches. These configuration parameters are generated using an off-line scheduler as explained in section 6.5. These parameters include node reserved resources for messages (i.e. time-triggered and rate-constraint) within the constituent system or border messages that are exchanged between the constituent systems. The configuration parameters contain schedules of several applications which emulate the real time SoS applications. The communication schedules are extended whenever additional scheduled messages are required. In order to achieve this, the incremental approach for deterministic networks should not require global knowledge about the overall network topology. The incremental scheduling is driven by the dependencies imposed by the DAG of an application. An application subsystem can be scheduled after the relied upon subsystems have been scheduled.

7.2 Tool Chain

The SoS simulation framework involves multiple tasks to be executed in series which can be described as a tool chain. Figure 7.1 illustrates the processes and shows the results after each stage of the chain. The first process is responsible for generating scenarios in terms of SoS topology as well as applications to be simulated. This data is fed to a scheduler process which performs incremental scheduling of each application based on the physical platform and generates incremental schedules. The third step is executed by a post processor which takes the generated schedules from the scheduler in addition to the physical and logical models taken from the generator and produces node configuration files suitable for OPNET simulation. The last process is the OPNET simulation execution and results generation. These processes are described in details in the following.

7.2.1 Post-Processing

The previously described scheduler generates schedules for all applications based on the physical SoS topology. These schedules include allocating jobs to end-systems, scheduling local and global messages to paths, and providing timely schedules for time triggered messages. However, the generated schedules do not include port assignments for the switches in the constituent systems as well as in the interaction domains. Moreover, the OPNET simulation works on configuration files which are used to setup the resources for each node. Thus, the schedules generated by the scheduler need to be processed first before the OPNET

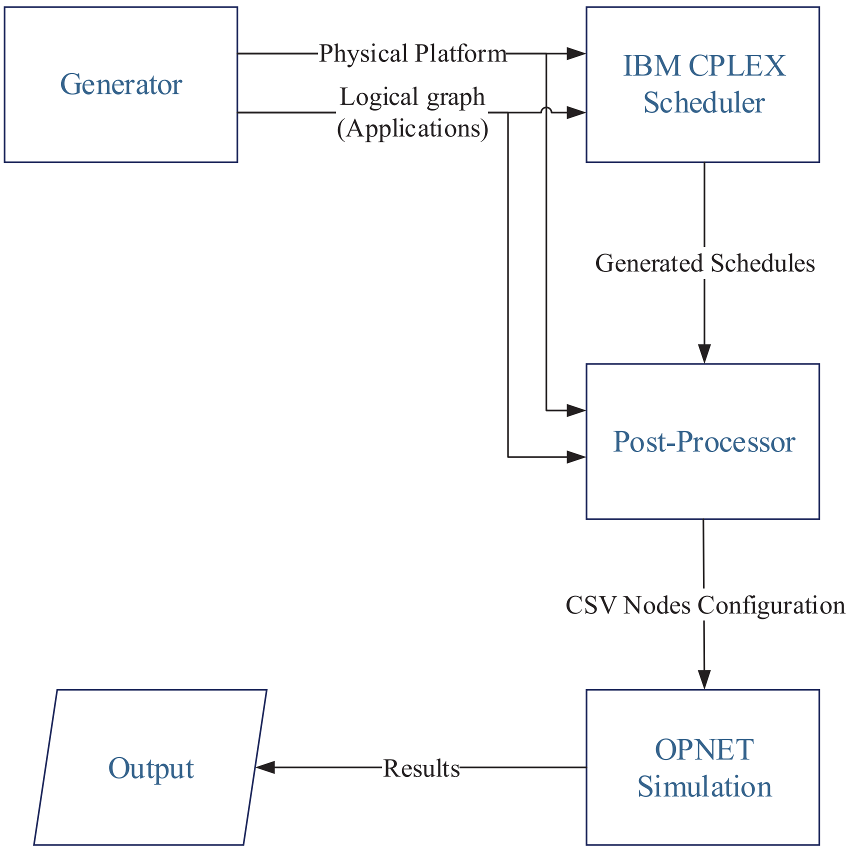


Fig. 7.1 Simulation Chain

simulation starts. The data produced in this stage consist of a trace of the different schedules which are used as an input for the simulation.

7.2.2 OPNET Simulation

The last phase in the tool chain is the OPNET simulation execution process and results generation. First of all, the physical network topology including node connectivity and port assignments need to be specified using the physical platform produced by the network generator. Then, these nodes (end-systems and switches) are configured with the configuration files generated by the post processor described above. After execution finishes, the OPNET simulation checks the correctness of the generated schedules by keeping track of packet collisions and drops during the whole period of the scenario execution and produces the results regarding the worst-case latency for time-triggered and rate-constrained messages.

7.3 Coordination Protocol of Time-Triggered Scheduling in SoS

The previous section demonstrates the required steps to update the configuration files of the SoS nodes using a tool chain. However, the schedules for the interested applications must be manually generated and then deployed to the related nodes before the simulation starts.

An SoS application can be scheduled where the CSMs in the constituent systems communicate with each other while generating their distributed schedules and then deploying the resulting schedule. This section presents the rules and mechanisms that are used between the constituent systems of the SoS. It defines a list of steps required whenever a new application is introduced. Moreover, it includes the control and the operational interfaces between the building blocks (end-systems, switches, and CSMs).

7.3.1 Message-based Interfaces between Building Blocks

Figure 7.2 illustrates the integration of the scheduler layer inside the block diagram of a TTEthernet end-system [AO13]. Besides the generic source that generates all traffic types (i.e., time-triggered, rate-constrained and best-effort), a scheduler is added to accept new application requests, communicate with other CSMs to generate the incremental schedules, and configures the nodes inside its constituent system with the generated schedule. The configuration messages sent and received by the scheduler layer are of best-effort type.

The job of the scheduler layer is summarized in Algorithm 2. The interested node sends a new application request to the broker of the CS, which forwards the request to the CSM with complete information about the high-level allocation of this application. After that, the root CSM sends this information to all CSMs related to the participating application subsystems which are allocated to constituent systems and waits for their acknowledgement messages. Moreover, the root CSM starts generating low-level allocation and scheduling of the jobs and messages within its application subsystem. After these CSMs received the application dependency information and send back the acknowledgement to the root CSM, the latter starts sending the global messages to those CSMs. A CSM will perform low-level allocation and scheduling of its jobs and messages if and only if the times of all dependent global messages have arrived. When all CSMs received acknowledgments about the completion of schedule generations from all participating CSMs, they will configure the interested end-systems and switches within the constituent system.

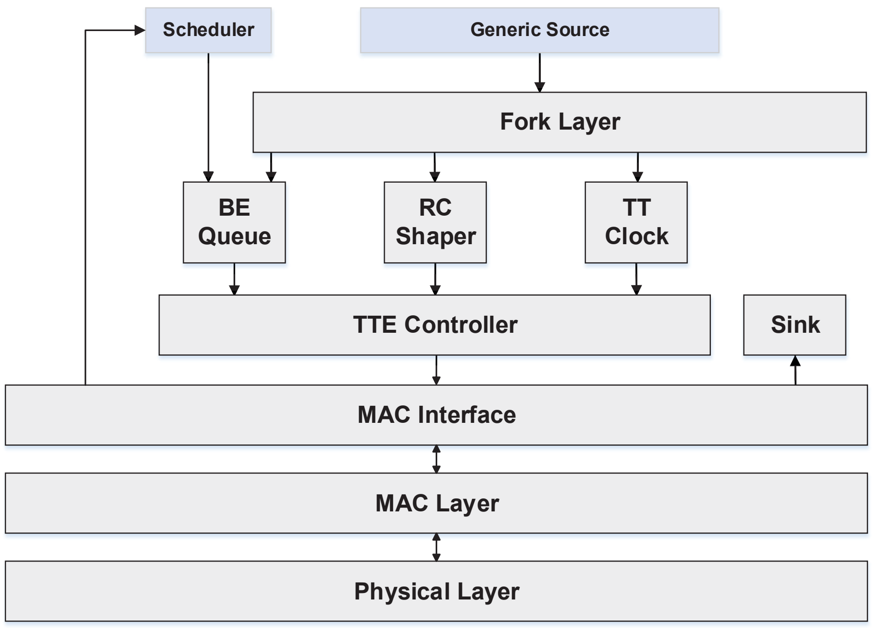


Fig. 7.2 Block diagram of CSM in TTEthernet End system

7.3.2 Sequence of Activities to Introduce a new Application

The interactions between the constituent system managers in the SoS, as well as, interactions between the requested application in a constituent system with its constituent system manager are classified into four main operations. The first operation consists of finding the high-level application subsystems of the application requests. The next operation is to perform a mapping of the high-level application subsystems to high-level physical models. This is followed by the scheduling of the low-level logical model in each mapped constituent system. Finally, the generated schedules are deployed in the selected physical models that necessitate the configuration of the end-systems in each constituent system by its constituent system manager CSM. The aforementioned operations can be explained in details using nine phases as depicted in figure 7.3.

Phase 0: A user, connected to an end-system, logs into the system and requests a service from the SoS. For example, in tele-health monitoring and patient service where a number of sensors are used to keep track of the status of a patient and send emergency requests in case of abnormal readings. The request is delivered to the operating system of the end-system.

Phase 1: An application in an end-system sends a request for a particular service to its CSM. It is assumed that any request made by an application includes information about the

```
switch Type do
    case Request to Root CSM
        Add new APP
        Send APP Dependency to all participating CSMs
        // perform low-level allocation and scheduling to $AS$ allocated to the root $CS$
        incremental update of $ALLOC_{\mathrm{job,c}}$ for jobs in $AS$
        incremental update of $\mathrm{SCHED}_{\mathrm{m,c}}$ for msgs. in $AS$
        break
    case Exchange of APP Dependency
        Send APP Dependency ACK to Root CS
        break
    case APP Dependency ACK
        Send Times of Dependent Msgs. to requester CSM
        break
    case Exchange Global Messages
        if ALL Dependent Msgs Received then
            // perform low-level allocation and scheduling to $AS$ allocated to this $CS$
            incremental update of $ALLOC_{\mathrm{job,c}}$ for jobs in $AS$
            incremental update of $\mathrm{SCHED}_{\mathrm{m,c}}$ for msgs. in $AS$
        end
        break
    case Schedule Generated ACK
        if ALL other Schedules of ASs Generated then
            Configure the PEs and SWs inside $CS$
        end
        break
    otherwise
        Do Nothing
    end
endsw
```

Algorithm 2: Interaction of the Scheduler layer to configuration messages

specific application subsystems, i.e., high-level logical models, which serve the intended request.

Phase 2: In this phase the CSM then consults the broker in the constituent system in order to procure the suitable constituent systems, i.e., high-level physical models, and perform a high-level allocation for the specific application subsystems defined in the request.

Phase 3: The CSM performs preliminary calculations to obtain the predecessor- and successor constituent systems for each of the involved constituent systems, i.e., high-level application dependencies.

Phase 4: In phase 4 the root CSM sends messages to all CSMs of the involved constituent systems about the high-level-application dependencies. Thus, every CSM is aware when to start its low-level scheduling.

Phase 5: The CSM performs a low-level scheduling algorithm. This includes the allocation of local jobs to end-systems as well as computing the optimal paths for all local and global messages of the first application subsystem.

Phase 6: When all CSMs receive a high-level-application dependency information message, they send back to the root CSM an acknowledgement message. This message confirms that all CSMs of successor constituent systems are ready to receive messages about finish time of high-level dependent messages.

Phase 7: In phase 6 the finish times of the global messages are sent to the CSMs of successor constituent systems. This information is used as an input, in addition to its low-level logical model, for the received CSMs in order to start their low-level scheduling algorithm.

Phase 8: When the CSMs of the successor constituent systems receive the predecessor message, they start computing their low-level schedule.

Phase 9: After the CSMs of the successor constituent systems generate the schedule, they hand in the results of their global messages to their successors' constituent system.

Phase 10: When the CSMs of the last successors receive the predecessor message, they compute their low-level schedule.

Phase 11: When the CSMs of the last successor constituent systems have finished with the generation of their low-level schedules, they send their global messages to the first CSM. This message indicates the end of the incremental scheduling for all involved constituent systems of the request.

Phase 12: The last phase is concerned with the deployment procedure of the request with the actual reconfiguration of the interested constituent systems based on the generated schedules.

Figure 7.4 illustrates the timing diagram of the application request management messages that are sent to schedule the application subsystems and to reconfigure the related constituent systems' nodes, namely end-systems and switches. The deployment of a new application schedule to the interested nodes is executed in the next global period in order to ensure its correct execution and to avoid negative interference with the running applications.

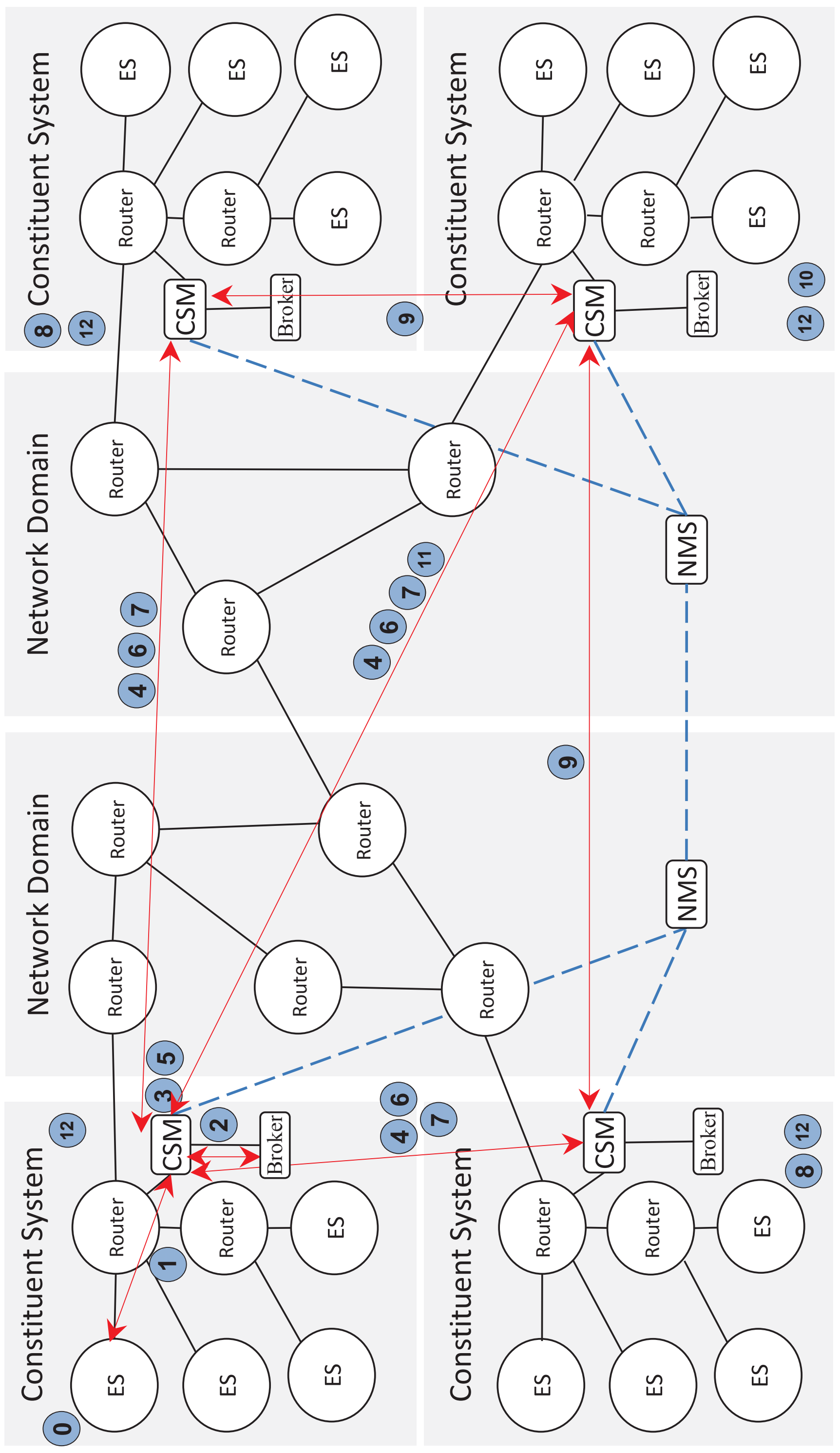


Fig. 7.3 Different phases of requests managements in SoS

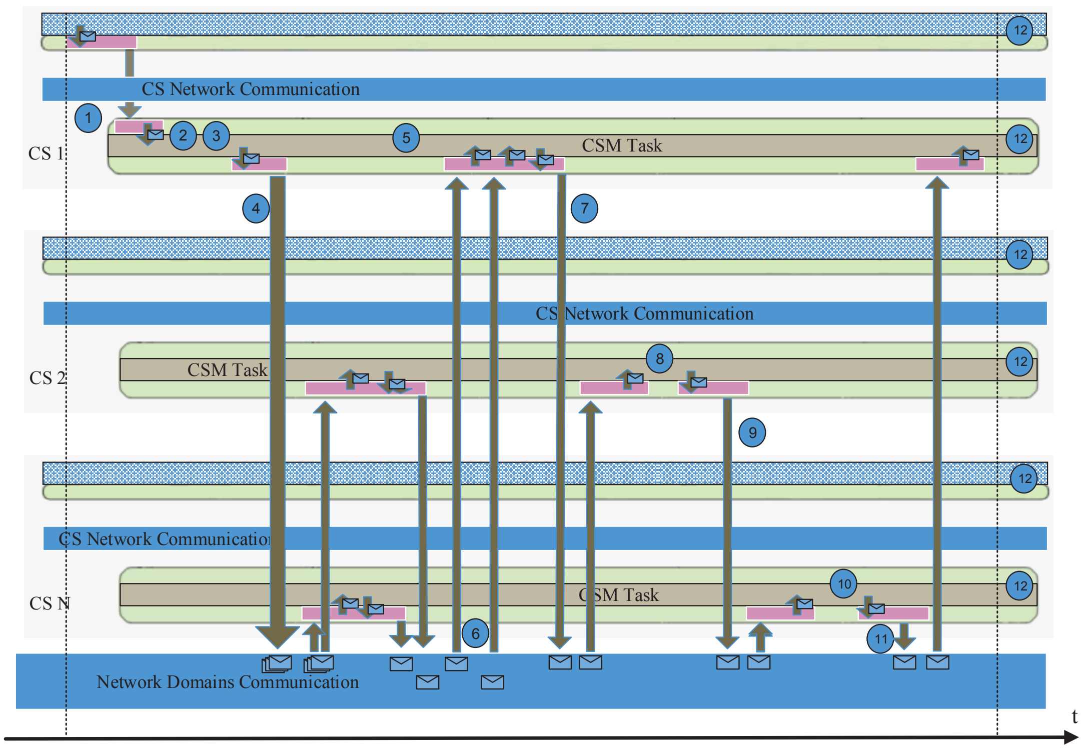


Fig. 7.4 Message sequence diagram

Figure 7.5 depicts the description of the messages transferred when a request is made by an end-system.

The *Service Request* message is an Ethernet message initiated by the requester end-system. It consists of the ID of the request and the high-level logical models that will serve the intended request. The logical models contains a vector of the required application subsystems and the adjacency matrix for the dependencies between the application subsystems.

The *Exchange of Dependencies* message is an Ethernet message that is sent by the base CSM to the CSMs of the selected constituent systems. It consists of the ID of the request and a unique application number which is a combination of the ID of the requester end-system and a sequence number generated by the CSM. Moreover, it contains the logical models and the selected constituent systems that will allocate the application subsystems.

The *Exchange of FinishTime* message is an Ethernet message that is sent by the CSMs of the predecessor constituent systems. In addition to the request ID and the application ID, this message consists of the sender constituent system's ID and the finish time of the global messages that are sent by this constituent system.

However, there are a number of assumptions to be considered in the proposed model.

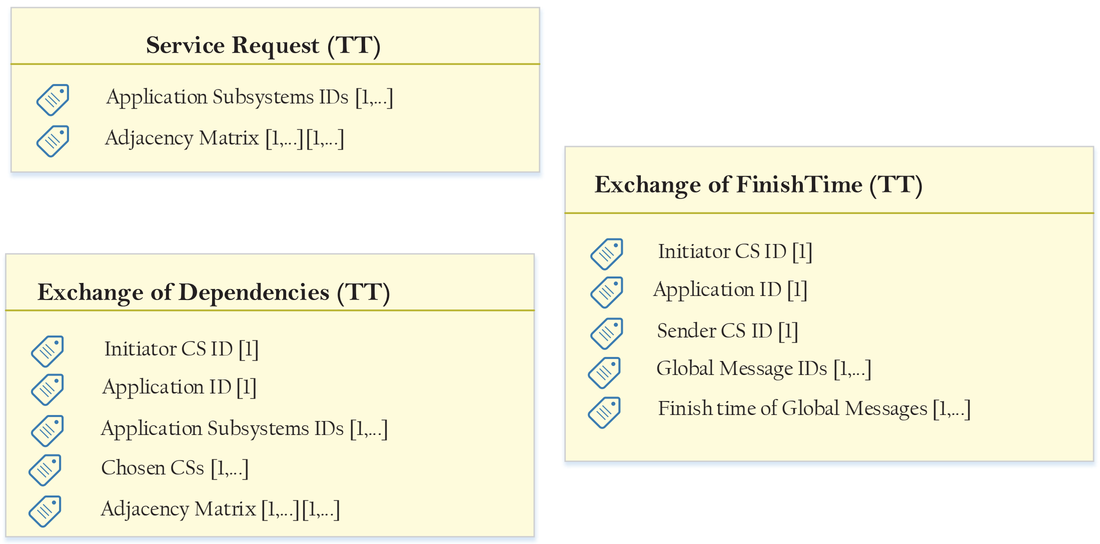

Service Request (TT)
Application Subsystems IDs [1,...]
Adjacency Matrix [1,...][1,...]
Exchange of FinishTime (TT)
Initiator CS ID [1]
Application ID [1]
Sender CS ID [1]
Global Message IDs [1,...]
Finish time of Global Messages [1,...]
Exchange of Dependencies (TT)
Initiator CS ID [1]
Application ID [1]
Application Subsystems IDs [1,...]
Chosen CSs [1,...]
Adjacency Matrix [1,...][1,...]

Fig. 7.5 Description of CSM messages

- Each Constituent System has a unique ID called CS-id. This ID is used for mapping the high-level application subsystems to constituent systems.
- All CSMs have clear information on the required maximum computation time of each job (WCET) where this value is constant.
- All CSMs have clear information on link maximum transmission time of a message (Duri) and it is a constant.
- The broker is implemented inside the CSM. Every CSM has a database with logical models that are used to serve requests. Depending on the type of a request the CSM looks-up a suitable logical model which is used as an input for local scheduling.

7.4 Experimental Evaluation

This section discusses the experimental evaluation of the implemented simulation using input scenarios. These scenarios are generated using the tool chain described in the previous section. Each scenario consists of a number of applications and each application consists of a number of time-triggered and rate-constrained messages that are scheduled to the constituent systems of the physical platform of the SoS. The SoS physical topology consists of 15 switches and 60 end-systems in the constituent systems in addition to 4 networked switches in the network domain as depicted in figure 7.6.

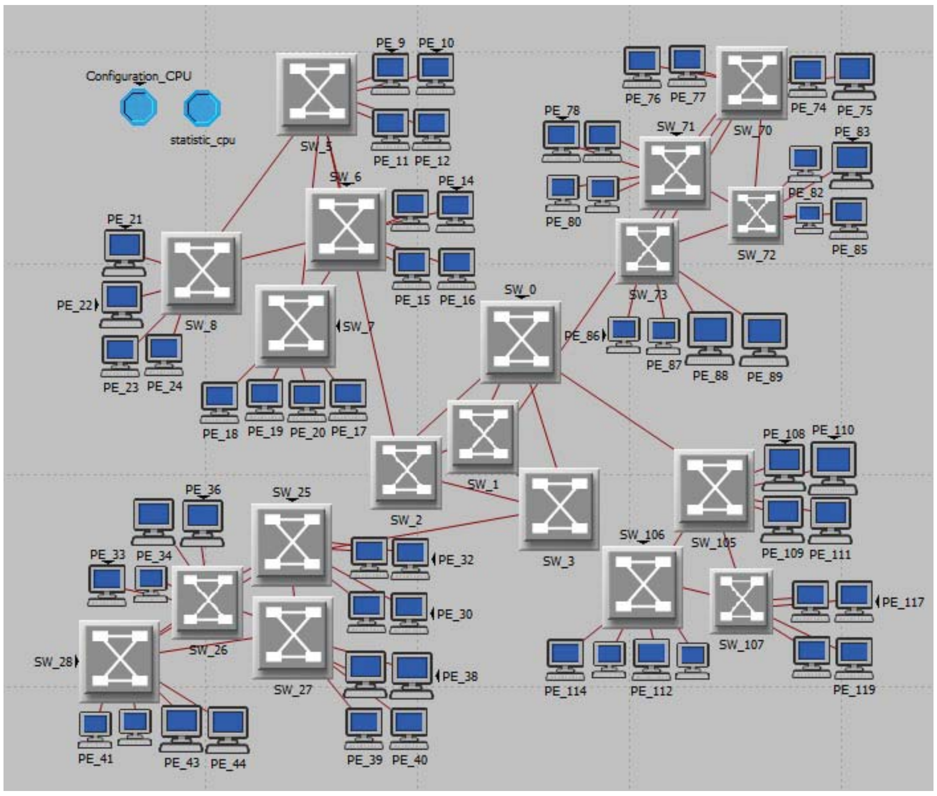


Fig. 7.6 Example of SoS with 4 CSs

No. of cases	WC latency of TT Msgs	WC latency of RC Msgs
5	0.017 ms	0.041 ms

Table 7.2 Use case result in OPNET simulation environment

The time-triggered messages are executed periodically and the rate-constrained messages are executed sporadically. The SNAP library is responsible for building the dependencies between these local as well as border messages. The schedules which are generated using CPLEX are the optimal solutions for the MILP problem explained in section 6.5. A total of 5 different logical models are generated and their schedules have been collected. Table 7.2 shows the results of the simulation environment. It summarizes the results of the 5 logical scenarios with the observed worst-case (WC) latencies for both time-triggered as well as rate-constrained messages.

To test the effect of faulty nodes on the schedule, consider an emergency room system in a hospital as shown in figure 7.7. The system provides medical diagnosis and treatment in case of emergency situations. A number of medical diagnosis sensors, such as Electrocardiography (ECG) (which is a time-triggered traffic) and blood pressure and motion detection (which are rate-constrained traffic) are placed in each patient's room. These sensors record the physical parameters of the patient and send them to a data acquisition system for data collection. These statistics are digitally tested by a pre-analyzer to check for abnormal conditions and then critical cases are sent to the central health administrator as well as to the central medical server [BDM14]. The administration section of the hospital gathers the patient's statistics along with the file of the patient and directs them to a general physician. Finally, a required medical expert is chosen to check the patient.

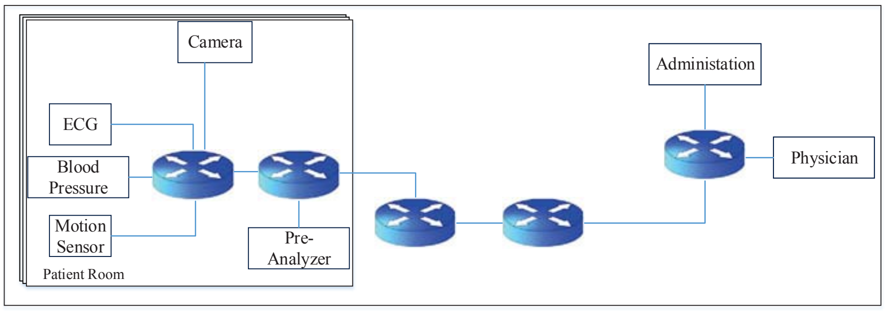
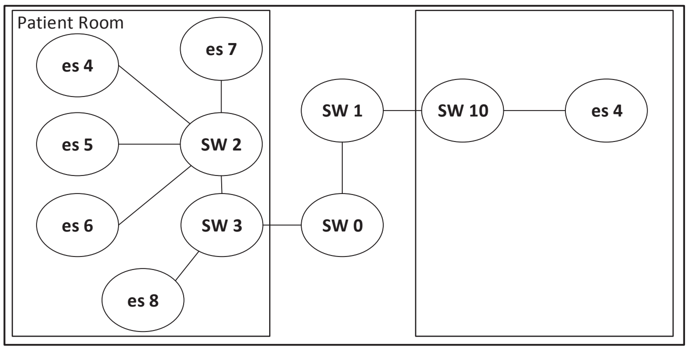


Fig. 7.7 Medical Scenario Use Case

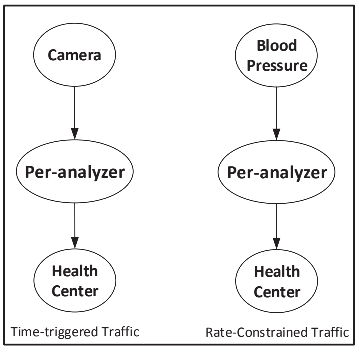
Figure 7.8 illustrates the physical as well as the logical models for the medical scenario. Table 7.3 lists the traffic generated by the running jobs in which they are statically allocated to nodes. The ECG sampling traffic is transmitted as time-triggered messages with a rate of 200 samples per second [NHRRT+17]. The allocated end-system to the ECG job reads an input file that contains temporal heartbeat information and transmits them to the pre-analyzer. The blood pressure job is transmitted as rate-constrained messages in $5msec$ intervals. Camera service transmits compressed video frames with a rate of $512Kb$ per second in which a packet of size 512 bits is transmitted every $1msec$.

The evaluation scenario consists of two parts, a scenario where the ECG sensor data are sent as a time-triggered traffic in a fault-free network and then with the introduction of a Bubbling Idiot (BI) failure as the video camera becomes faulty and starts flooding the network with messages. The behavior of the time-triggered messages of the ECG output is not affected by the camera failure. The simulated behavior of the ECG outputs is shown in figure 7.9.

The second scenario where the ECG sensor data are sent as a time-triggered traffic in a fault-free network and then with the introduction of a BI failure. The behavior of the rate-constraints messages in the ECG output is affected by the camera failure in which the shape of ECG output is affected as depicted in figure 7.10.



(a) Physical Model



(b) TT and RC Logical Model

Fig. 7.8 Model example for medical scenario

Job Name	Assgned end-system	Period/BAG	Traffic Type
ECG	4	5 ms	TT
Blood pressure	5	5 s	RC
Pre-analyzer (ECG)	8	5 ms	TT
Pre-analyzer (other)	8	10 s	RC
Camera	7	1 ms	BE

Table 7.3 Characteristics and assignment of jobs to End-systems

The simulation framework for safety critical applications in an SoS is an excellent tool to evaluate and validate schedule results of SoS applications. The simulation chain creates physical platform and logical application models for an SoS that are scheduled using incremental, distributed, and concurrent scheduler. The provided schedule is fed into the proposed SoS real-time Ethernet simulation framework after pre-processing of the scheduled applications (i.e., temporal switch port assignments, virtual link generations, and end-system injection time schedules) to generate the OPNET configuration files for both end-systems and switches.

The SoS simulation framework provides feedback on the temporal behavior of the SoS nodes, namely TTEthernet end-systems and switches. It keeps track of packet collisions and drops during the simulation run-time, records end-to-end latency for time-triggered as well as rate-constrained messages, and analyze the received messages in the application level.

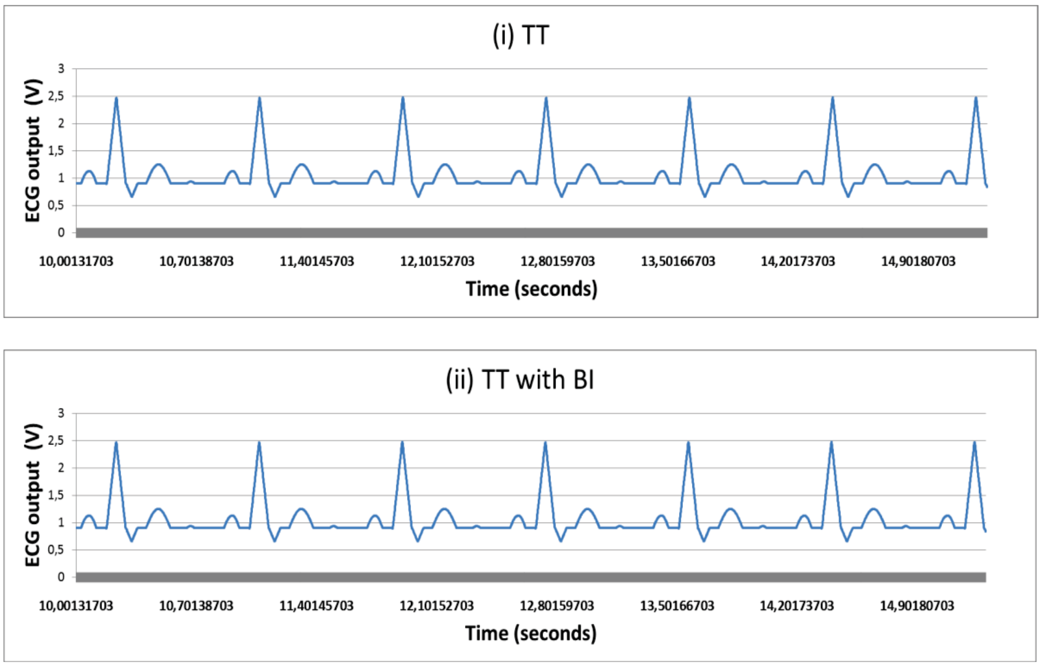


Fig. 7.9 Effect of BI for TT

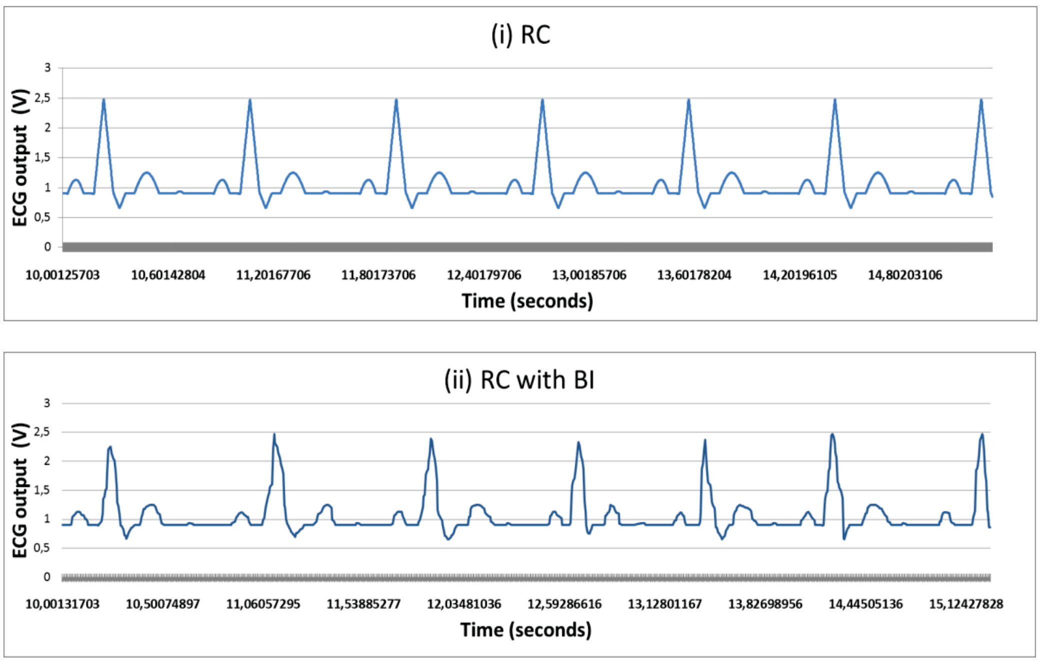


Fig. 7.10 Effect of BI for RC

Chapter 8

Conclusion

The field of embedded systems based on multi-core processors is facing new challenges due to applications with mixed-criticality. Moreover, the increasing importance of Systems-of-Systems (SoSs) with real-time requirements demands techniques for scheduled end-to-end communication with resource reservations and temporal guarantees. This dissertation introduced a scheduling model for mixed-criticality systems based on multi-core processors that supports different traffic types and fault tolerance. The scheduling model was designed first to work at the level of local networks and then was extended to support Systems-of-Systems (SoSs). The formulation of these models was based on Mixed Integer Linear Programming (MILP) and their example scenarios are solved optimally using the IBM CPLEX optimizer.

At the level of a single network of multi-core processors the scheduling model was designed to support both event-triggered and time-triggered multi-hop communication with precedence constraints. The combination of these two timing models is significant in many applications, where regular subsystems (e.g., control functions, alarm monitoring) coexist with sporadic subsystems (e.g., user interfaces, event-driven activities). The selected message paths avoid collisions between time-triggered messages and consider the utilization of switches for event-triggered messages. The proposed model minimizes the latencies and provides an optimal allocation and schedule of the computational jobs and communication activities.

A spatial redundancy technique was integrated in the proposed model to guarantee the timely delivery of time-triggered messages in the presence of communication link faults. The implemented model maximizes the reliability of time-triggered messages using redundant paths. The proposed model generates redundant schedules for time-triggered messages while taking into account message dependencies, collision-avoidance and real-time constraints of message transmissions.

The scheduling of SoS requires that the end-to-end communication channels need to span constituent systems with operational and managerial independence, while also supporting an evolutionary development and the dynamic introduction of new applications. The proposed scheduling model solves this challenge by performing an incremental, distributed and concurrent scheduling of applications. Each constituent system and each network domain is equipped with management services for incrementally computing local schedules for the respective application subsystems. The resulting timing information of relied upon messages is exchanged between constituent systems to satisfy the temporal dependencies between application subsystems. The distributed scheduling reduces the overall computation time by splitting the scheduling problem of the whole SoS network into a number of sub-networks (constituent systems) and at the same time solves the problem of independent management and lack of centralized global knowledge about the internals of constituent systems. The evaluation demonstrates the achievable real-time guarantees and the computation time for scheduling based on example scenarios.

The thesis also introduced an SoS simulation and verification environment which supports real-time requirements. The simulation environment performs automatic generation of random application scenarios consisting of time-triggered and rate-constrained messages which are solved used the previously proposed SoS scheduling model. The generated schedules are then fed into the simulation. The simulation environment verifies the correctness of the generated schedules and evaluates the worst-case message latencies. Moreover, the importance of time-triggered messages for critical applications is verified by introducing a babbling Idiot (BI) failure where the behavior of the service when scheduled as a time-triggered message is not affected compared to a rate-constraint message. Moreover, generic simulation building blocks for SoS were implemented and integrated with the scheduling model to allow the generation and verification of example scenarios at run-time.

References

[ABD$^+$13] Andreas Abel, Florian Benz, Johannes Doerfert, Barbara Dörr, Sebastian Hahn, Florian Haupenthal, Michael Jacobs, AmirH Moin, Jan Reineke, Bernhard Schommer, and Reinhard Wilhelm. Impact of resource sharing on performance and performance prediction: A survey. In P.R. D'Argenio and H. Melgratti, editors, *CONCUR 2013 – Concurrency Theory*, volume 8052 of *Lecture Notes in Computer Science*, pages 25–43. Springer Berlin Heidelberg, 2013.

[ACP04] Giuseppe Ascia, Vincenzo Catania, and Maurizio Palesi. Multi-objective mapping for mesh-based noc architectures. In *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 182–187. ACM, 2004.

[AGK12] Ismail Assayad, Alain Girault, and Hamoudi Kalla. Scheduling of real-time embedded systems under reliability and power constraints. In *Complex Systems (ICCS), 2012 International Conference on*, pages 1–6. IEEE, 2012.

[AGRN16] Guy Avni, Shibashis Guha, and Guillermo Rodriguez-Navas. Synthesizing time-triggered schedules for switched networks with faulty links. In *Embedded Software (EMSOFT), 2016 International Conference on*, pages 1–10. IEEE, 2016.

[AHU] Alfred V Aho, John E Hopcroft, and Jeffrey D Ullman. Data structures and algorithms, 1983.

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

[ALRL04] Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1):11–33, 2004.

[AO13] M. Abuteir and R. Obermaisser. Simulation environment for time-triggered ethernet. In *Industrial Informatics (INDIN), 2013 11th IEEE International Conference on*, pages 642–648, July 2013.

[AS05] Alper Atamtürk and Martin WP Savelsbergh. Integer-programming software systems. *Annals of operations research*, 140(1):67–124, 2005.

[AS611] SAE AS6802. Time-triggered ethernet. *SAE International*, 2011.

[Avi76] Algirdas Avizienis. Fault-tolerant systems. *IEEE Trans. Computers*, 25(12):1304–1312, 1976.

[BAB96] Doug Burger, Todd M Austin, and Steve Bennett. *Evaluating future microprocessors: The simplescalar tool set*. University of Wisconsin-Madison, Computer Sciences Department, 1996.

[Baz92] M Bazarra. Linear programming and network flows, 1992.

[BC11] S. Borkar and A.A. Chien. The future of microprocessors. *Communications of the ACM*, 54(5):67–77, 2011.

[BCDV91] Ricky W Butler, James L Caldwell, and Ben L Di Vito. Design strategy for a formally verified reliable computing platform. In *Computer Assurance, 1991. COMPASS'91, Systems Integrity, Software Safety and Process Security. Proceedings of the Sixth Annual Conference on*, pages 125–133. IEEE, 1991.

[BDM14] S. S. Bhunia, S. K. Dhar, and N. Mukherjee. ihealth: A fuzzy approach for provisioning intelligent health-care system in smart city. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2014 IEEE 10th International Conference on*, pages 187–193, Oct 2014.

[BK00] Günther Bauer and Hermann Kopetz. Transparent redundancy in the time-triggered architecture. In *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on*, pages 5–13. IEEE, 2000.

[Bow00] Jonathan Bowen. The ethics of safety-critical systems. *Commun. ACM*, 43(4):91–97, April 2000.

[BS06] John Boardman and Brian Sauser. System of systems-the meaning of of. In *System of Systems Engineering, 2006 IEEE/SMC International Conference on*, pages 6–pp. IEEE, 2006.

[BS09] W Clifton Baldwin and Brian Sauser. Modeling the characteristics of system of systems. In *System of Systems Engineering, 2009. SoSE 2009. IEEE International Conference on*, pages 1–6. IEEE, 2009.

[BTM00] David Brooks, Vivek Tiwari, and Margaret Martonosi. *Wattch: A framework for architectural-level power analysis and optimizations*, volume 28. ACM, 2000.

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

[CASTCA16] Position Paper Certification Authorities Software Team CAST-32A, FAA. *Multi-core Processors*, May 2016.

[CDK05] George F Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed systems: concepts and design*. pearson education, 2005.

[CLM$^+$04] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra. Spidergon: a novel on-chip communication network. In *System-on-Chip, 2004. Proceedings. 2004 International Symposium on*, page 15, 2004.

[CO14] Silviu S Craciunas and Ramon Serna 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*, page 45. ACM, 2014.

[Dav13] R. Davis. *Static Probabilistic Timing Analysis for Multicore Processors with Shared Cache*, pages 3–5. 2013.

[DT03] William James Dally and Brian Patrick Towles. *Principles and practices of interconnection networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[Dub13] Elena Dubrova. *Fault-tolerant design*. Springer, 2013.

[DYN03a] Jose Duato, Sudhakar Yalamanchili, and Lionel M Ni. *Interconnection networks: an engineering approach*. Elsevier, 2003.

[DYN03b] José Duato, Sudhakar Yalamanchili, and Lionel M. Ni. *Interconnection networks: An engineering approach*. Morgan Kaufmann, San Francisco, CA, rev. print edition, 2003.

[Edi12] Editted by: Roman Obermaisser. *Time-Triggered Communication*. Embedded Systems. CRC Press, USA, 2012.

[Far06] Emilia Farcas. *Scheduling multi-mode real-time distributed components*. PhD thesis, PhD thesis, Department of Computer Sciences, University of Salzburg, 2006.

[FIW15] FIWARE. *IoT Discovery – User and Programmers Guide*, 2015.

[GAC$^+$13] Kees Goossens, Arnaldo Azevedo, Karthik Chandrasekar, Manil Dev Gomony, Sven Goossens, Martijn Koedam, Yonghui Li, Davit Mirzoyan, Anca Molnos, Ashkan Beyranvand Nejad, Andrew Nelson, and Shubhendu Sinha. Virtual execution platforms for mixed-time-criticality systems: The compsoc architecture and design flow. *SIGBED Rev.*, 10(3):23–34, October 2013.

[Gas88] Morrie Gasser. *Building a Secure Computer System*. Van Nostrand Reinhold Co., New York, NY, USA, 1988.

[GDR05] K. Goossens, J. Dielissen, and A. Radulescu. Aethereal network on chip: concepts, architectures, and implementations. *Design & Test of Computers, IEEE*, 22(5):414–421, 2005.

[GGLR98] Alex Gantman, Pei-Ning Guo, James Lewis, and Fakhruddin Rashid. Scheduling real-time tasks in distributed systems: A survey. 1998.

[GGS11] S Jimmy Gandhi, Alex Gorod, and Brian Sauser. A systemic approach to managing risks of sos. In *Systems Conference (SysCon), 2011 IEEE International*, pages 412–416. IEEE, 2011.

[GLH+11] Qiang Guo, Xuhong Li, Lipo Huang, Jianjun Gao, and Xia Xiao. Research on reliability optimization of weapon system based on heuristic arithmetic. In *2011 International Conference on System science, Engineering design and Manufacturing informatization*, volume 2, pages 24–26, Oct 2011.

[GV02] Tony Givargis and Frank Vahid. Platune: a tuning framework for system-on-a-chip platforms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(11):1317–1327, 2002.

[HS97] Chao-Ju Hou and Kang G. Shin. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. *IEEE transactions on computers*, 46(12):1338–1356, 1997.

[IEE15a] IEEE. *IEEE 802.1Qbv – Enhancements for Scheduled Traffic, Draft 2.4*, 2015.

[IEE15b] IEEE. Ieee 802.1qbv. enhancements for scheduled traffic. Draft 3.1, 2015.

[ISO15] Iso/iec/ieee international standard - systems and software engineering – system life cycle processes. *ISO/IEC/IEEE 15288 First edition 2015-05-15*, pages 1–118, May 2015.

[Jam08] Mo Jamshidi. *Systems of systems engineering: principles and applications*. CRC press, 2008.

[Jam09] M. Jamshidi. *Systems of Systems Engineering – Principles and Applications*. CRC Press, 2009.

[Jaz14] N. Jazdi. Cyber physical systems in the context of industry 4.0. In *Automation, Quality and Testing, Robotics, 2014 IEEE International Conference on*, pages 1–4, May 2014.

[Kam11] Raj Kamal. *Embedded systems: architecture, programming and design*. Tata McGraw-Hill Education, 2011.

[Kav92] Krishna M Kavi. *Real-Time Systems, Abstractions, Languages, Design Methodologies, and Tools*. IEEE Computer Society Press, 1992.

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

[KQA+14] L. Kosmidis, E. Quinones, J. Abella, T. Vardanega, I. Broster, and F.J. Cazorla. Measurement-based probabilistic timing analysis and its impact on processor architecture. In *Digital System Design (DSD), 2014 17th Euromicro Conference on*, pages 401–410, Aug 2014.

[Les] J. Leskovec. Stanford Network Analysis Package(SNAP). http://snap.stanford.edu/. [Online; accessed 26-February-2015].

[LG11] Ricardo M Lima and Ignacio E Grossmann. Computational advances in solving mixed integer linear programming problems. 2011.

[LH94] Jaynarayan H Lala and Richard E Harper. Architectural principles for safety-critical real-time applications. *Proceedings of the IEEE*, 82(1):25–40, 1994.

[LH98] Yanbing Li and Jörg Henkel. A framework for estimation and minimizing energy dissipation of embedded hw/sw systems. In *Proceedings of the 35th annual Design Automation Conference*, pages 188–193. ACM, 1998.

[LS16] Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Trans. Intell. Syst. Technol.*, 8(1):1:1–1:20, July 2016.

[M. 09] M. Coppola, M. D. Grammatikakis, R. Locatelli, G. Maruccia, L. Pieralisi. *Design of Cost-efficient Interconnect Processing Units: Spidergon STNoC*. CRC Press, USA, 2009.

[Mai98] M.W. Maier. *Architecting Principles for Systems-of-Systems – Systems Engineering*. 1998.

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

[MAO17] A. Murshed, M. Abuteir, and R. Obermaisser. Validation framework for time-triggered system-of-systems. In *Promising Electronic Technologies (ICPET), 2017 International Conference on*, pages 103–108. IEEE, 2017.

[MB06] G. de Micheli and L. Benini. *Networks on Chips: Technology and Tools*. Elsevier Science, 2006.

[MJ95] Joseph Mattai and Mathai Joseph. *Real-Time Systems: specification, verification, and analysis*. Prentice Hall PTR, 1995.

[MO17] A. Murshed and R. Obermaisser. Scheduler for reliable distributed systems with time-triggered networks. In *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, pages 425–430, July 2017.

[MOAK15] A. Murshed, R. Obermaisser, H. Ahmadian, and A. Khalifeh. Scheduling and allocation of time-triggered and event-triggered services for multi-core processors with networks-on-a-chip. In *Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on*, pages 1424–1431. IEEE, 2015.

[MTCM12] D. Mora, M. Taisch, A.W. Colombo, and J.M. Mendes. Service-oriented architecture approach for industrial system of systems: State-of-the-art for energy management. In *10th IEEE International Conference on Industrial Informatics (INDIN)*, pages 1246–1251, July 2012.

[MVPA13] Luis Marques, Verónica Vasconcelos, Paulo Pedreiras, and Luis Almeida. Error recovery in time-triggered communication systems using servers. In *Industrial Embedded Systems (SIES), 2013 8th IEEE International Symposium on*, pages 205–212. IEEE, 2013.

[Nel90] Victor P. Nelson. Fault-tolerant computing: Fundamental concepts. *Computer*, 23(7):19–25, 1990.

[NHRRT$^+$17] David Naranjo-Hernández, Laura M Roa, Javier Reina-Tosina, Gerardo Barbarov-Rostan, and Omar Galdámez-Cruz. Smart device for the determination of heart rate variability in real time. *Journal of Sensors*, 2017, 2017.

[NP12] J. Nowotsch and M. Paulitsch. Leveraging multi-core computing architectures in avionics. In *Dependable Computing Conference (EDCC), 2012 Ninth European*, pages 132–143, May 2012.

[NSL09] N Navet and F Sommot-Lion. Automotive embedded systems handbook. industrial information technological series, 2009.

[OKP10] R. Obermaisser, H. Kopetz, and C. Paukovits. A cross-domain multiprocessor system-on-a-chip for embedded real-time systems. *Industrial Informatics, IEEE Transactions on*, 6(4):548–567, Nov 2010.

[OM15] R. Obermaisser and A. Murshed. Incremental, distributed, and concurrent scheduling in systems-of-systems with real-time requirements. In *Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on*, pages 1918–1927. IEEE, 2015.

[OP06] Roman Obermaisser and Philipp Peti. A fault hypothesis for integrated architectures. In *Intelligent Solutions in Embedded Systems, 2006 International Workshop on*, pages 1–18. IEEE, 2006.

[OPN] OPNET Technologies. *OPNET Modeler 17.1 Documentation.*

[Pau08] Christian Peter Paukovits. *The time-triggered system-on-chip architecture: Vienna University of Technology, PhD thesis.* 2008.

[PG02] Maurizio Palesi and Tony Givargis. Multi-objective design space exploration using genetic algorithms. In *Hardware/Software Codesign, 2002. CODES 2002. Proceedings of the Tenth International Symposium on*, pages 67–72. IEEE, 2002.

[PSZ] Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria. Multi-objective design space exploration of embedded systems. *Journal of Embedded Computing*, 1(3).

[Ram90] Krithi Ramamritham. Allocation and scheduling of complex periodic tasks. In *Distributed Computing Systems, 1990. Proceedings., 10th International Conference on*, pages 108–115. IEEE, 1990.

[SAE] SAE International. *AS-6802 – Time-Triggered Ethernet.*

[San15] Imad Sanduka. *A modelling framework for systems-of-systems with real-time and reliability requirements.* PhD thesis, University of Siegen, 2015.

[SAS] SAS Institute Inc. *SAS/OR® 13.2 User's Guide: Mathematical Programming.*

[SCC$^+$12] D. Suparna, G. Cassar, B. Christophe, S.B. Fredj, M. Bauer, N. Santos, T. Jacobs, R. de las Heras, G. Martin, G. Völksen, and A. Ziller. Internet of things architecture. concepts and solutions for entity-based discovery of iot resoures and managing their dynamic associations. Technical report, 2012.

[Shi09] KV Shibu. *Introduction to Embedded Systems.* Tata McGraw-Hill Education, 2009.

[Sho01] Martin L Shooman. Reliability of computer systems and networks: Fault tolerance. *Analysis, and Design, Wiley-Interscience*, 2001.

[SJS07] Ferat Sahin, Mo Jamshidi, and Prasanna Sridhar. A discrete event xml based simulation framework for system of systems architectures. In *System of Systems Engineering, 2007. SoSE'07. IEEE International Conference on*, pages 1–7. IEEE, 2007.

[SKBMO15] C. Schöler, R. Krenz-Bååth, A. Murshed, and R. Obermaisser. Optimal sat-based scheduler for time-triggered networks-on-a-chip. In *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–6, Jun 2015.

[SKBMO16] C. Schöler, R. Krenz-Bååth, A. Murshed, and R. Obermaisser. Computing optimal communication schedules for time-triggered networks using an smt solver. In *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*, pages 1–9, May 2016.

[SMG00] Ricardo Sanz, Fernando Matía, and Santos Galán. Fridges, elephants, and the meaning of autonomy and intelligence. In *Intelligent Control, 2000. Proceedings of the 2000 IEEE International Symposium on*, pages 217–222. IEEE, 2000.

[Sta88] J. A. Stankovic. Misconceptions about real-time computing: a serious problem for next-generation systems. *Computer*, 21(10):10–19, Oct 1988.

[Tan95] Andrew S Tanenbaum. *Distributed operating systems.* Pearson Education India, 1995.

[UBG$^+$13] T. Ungerer, C. Bradatsch, M. Gerdes, F. Kluge, R. Jahr, J. Mische, J. Fernandes, P.G. Zaykov, Z. Petrov, B. Boddeker, S. Kehr, H. Regler, A. Hugl, C. Rochange, and Ozaktas. parmerasa - multi-core execution of parallelised hard real-time applications supporting analysability. In *DSD*, pages 363–370. IEEE, 2013.

[VKI$^+$00] Narayanan Vijaykrishnan, Mahmut Kandemir, Mary Jane Irwin, Hyun Suk Kim, and Wu Ye. Energy-driven integrated hardware-software optimizations using simplepower. *ACM SIGARCH Computer Architecture News*, 28(2):95–106, 2000.

[WCBM07] N. Wickramasinghe, S. Chalasani, R.V. Boppana, and A.M. Madni. Healthcare system of systems. In *System of Systems Engineering, 2007. SoSE '07. IEEE International Conference on*, pages 1–6, April 2007.

[WWP06] Marko Wolf, André Weimerskirch, and Christof Paar. Secure in-vehicle communication. *Embedded Security in Cars*, pages 95–109, 2006.

[ZBC$^+$14] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi. Internet of things for smart cities. *Internet of Things Journal, IEEE*, 1(1):22–32, Feb 2014.

[Zei90] Bernard P Zeigler. High autonomy systems: concepts and models. In *AI, Simulation and Planning in High Autonomy Systems, 1990., Proceedings.*, pages 2–7. IEEE, 1990.

[ZLT01] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm. 2001.

[Zur04] Richard Zurawski. *The industrial information technology handbook*. CRC Press, 2004.

Acronym

BAG	Bandwidth Allocation Gap
BI	Bubbling Idiot
COTS	commercial off-the-shelf
CSM	Constituent System Manager
DAG	Directed Acyclic Graph
ECG	Electrocardiography
EFP	Extra-functional Properties
FCR	Fault Containment Region
LP	Linear Programming
MILP	Mixed-Integer Linear Programming
MOST	Media Oriented System Transport
MPSoC	Multi-Processor-System-on-a-Chip
NI	Network Interface
NMS	Network Management Systems
NoC	Network-on-Chip
SMT	Satisfiability Modulo Theories
SNAP	Stanford Network Analysis Platform
SoC	Socket-on-Chip

SoS	System-of-Systems
TDMA	Time-division Multiple Access
TSN	Time Sensitive Networking
TTEthernet	Time-Triggered Ethernet
TTNoC	Time-Triggered Network-on-Chip
VL	Virtual Link
WCCOM	Worst-Case Communication Time
WCET	Worst-Case Execution Time

List of Publication

1. A. Murshed, R. Obermaisser, H. Ahmadian, and A. Khalifeh. Scheduling and allocation of time-triggered and event-triggered services for multi-core processors with networks-on-a-chip. In *Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on*, pages 1424–1431. IEEE, 2015

2. R. Obermaisser and A. Murshed. Incremental, distributed, and concurrent scheduling in systems-of-systems with real-time requirements. In *Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on*, pages 1918–1927. IEEE, 2015

3. A. Murshed, M. Abuteir, and R. Obermaisser. Validation framework for time-triggered system-of-systems. In *Promising Electronic Technologies (ICPET), 2017 International Conference on*, pages 103–108. IEEE, 2017

4. A. Murshed and R. Obermaisser. Scheduler for reliable distributed systems with time-triggered networks. In *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, pages 425–430, July 2017

5. C. Schöler, R. Krenz-Bååth, A. Murshed, and R. Obermaisser. Optimal sat-based scheduler for time-triggered networks-on-a-chip. In *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–6, Jun 2015

6. C. Schöler, R. Krenz-Bååth, A. Murshed, and R. Obermaisser. Computing optimal communication schedules for time-triggered networks using an smt solver. In *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*, pages 1–9, May 2016