Analysis methods for mixed-criticality applications on TTEthernet-based distributed architectures

Sorin Ovidiu Marinescu

Kongens Lyngby 2012
IMM-MSc-2012-139

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-MSc-2012-139

Abstract

More and more, we can see how embedded systems are given great responsibility in applications where their malfunctioning could have catastrophic effects. Often, embedded systems are used in areas where timing constraints need to be satisfied. For the hard real-time embedded systems it is important not only to produce the correct computation result, but also to deliver it at the correct moment in time. It's common, due to physical, safety or modularity constraints, to have these embedded systems organized in distributed architectures.

The initial approach used in the automotive and avionics fields for safety-critical real-time applications was to have each function running on a dedicated hardware node, with the nodes being interconnected in a network. This was the so-called "federated architecture". In time, the number of nodes in this federated distributed architecture has significantly increased, reaching more than one hundred in a modern car. This has led to increased power consumption, wiring, size, weight and costs.

The solution to the problems raised by the federated architectures was the so-called "integrated architecture", where multiple functions were allowed to share one node. Furthermore, the current trends are towards mixed-criticality systems, where functions with different criticalities can coexist onto the same node. In avionics, this integration is realized, at CPU-level, on the basis of "Integrated Modular Avionics" (IMA), also known as the ARINC 653 standard. IMA specifies the rules for allowing the integration of mixed-criticality functions onto the same node.

The concept of integrated architecture implies that the different criticality functions also share the same communication network, not just the same computing

platform. At network level, one of the communication protocols for applications of mixed-criticality is TTEthernet. Based on Ethernet, TTEthernet supports both time-triggered and event-triggered traffic and provides both spatial and temporal separation between these two traffic classes and is, thus, suitable for mixed-criticality applications.

The objective of the thesis is to propose and develop analysis methods for mixed-criticality applications on TTEthernet-based distributed architectures. We have extended the state-of-the-art schedulability analysis for tasks to take into account partitions at the CPU-level. For TTEthernet, we have designed and implemented a simulator, whose results were compared to the results provided by a previously proposed TTEthernet analysis. We have also proposed a new TTEthernet analysis, which is based on extending the so-called "trajectory approach" used for the analysis of Avionics Full-Duplex Switched Ethernet, a precursor of TTEthernet. The proposed methods have been implemented in a tool, which has been evaluated using several synthetic and realistic benchmarks. Analysis tools are needed to support the designer in obtaining schedulable and cost-effective implementations of mixed-criticality applications on partitioned architectures.

Acknowledgments

I wish to express my gratitude towards Paul Pop for being a great supervisor. He helped me shape this project and he encouraged me when I was stuck. His feedback was always concise, precise and valuable.

I would also like to say a big thank you to Domiţian Tămaş–Selicean for our fruitful discussions, for his help with the SA analysis for IMA and with the TTEthernet analysis and for always answering my emails even when he was very busy on a different continent.

Contents

Chapter 1

Introduction

The computing systems have had a spectacular evolution. From the big mainframe computers from the 60s until the driverless cars of today, they have become cheaper, more accessible and took an increasingly important role in our lives. The embedded systems have a central part in the story of computing systems. The embedded systems are computers usually with a single purpose, designed to fit their application's specific requirements. We can find them everywhere - in digital watches, as well as in nuclear reactors.

With such a broad utilization, the embedded systems are very varied. From the perspective of our thesis, we can distinguish the category of *hard real-time* embedded systems in which it is not enough for a system to produce a correct computation result, but it is also needed that the result is obtained at the correct moment in time [Kop11]. In this context, the issue of safety (understood as a property of a system that it will provide a hazard-free operation and, as a consequence, not endanger the human life or the environment) is raised. The *safety-critical* embedded systems are another category of interest for us. The recent years saw the emergence of *mixed-criticality* systems. A mixed-critical system is defined in the research agenda for mixed-criticality systems [BBB+09] as "an integrated suite of hardware, operating system and middleware services and application software that supports the execution of safety-critical, mission-critical, and non-critical software within a single, secure compute platform". The mixed-criticality distributed embedded systems constitute the focus of our

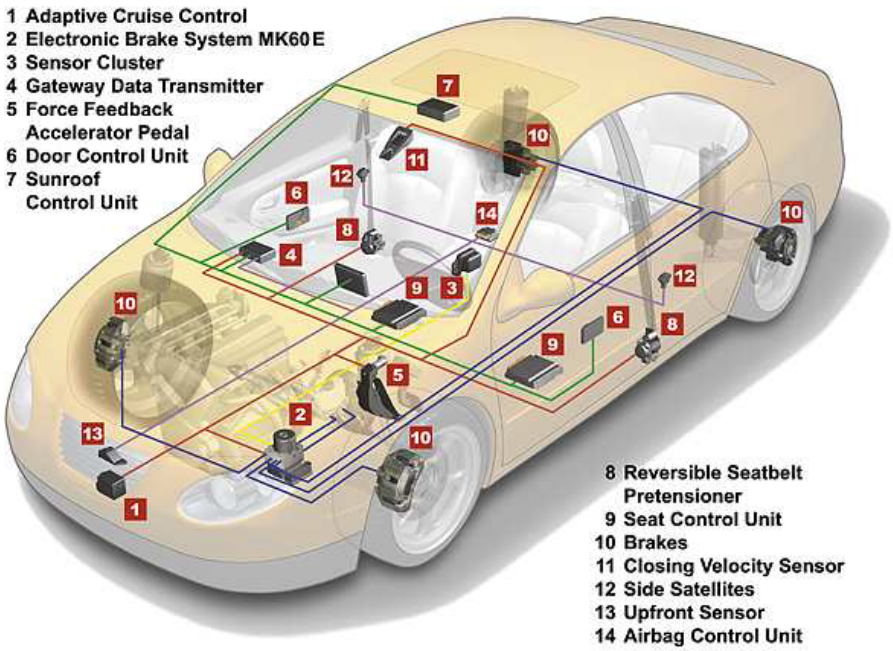


Figure 1.1: Example of a distributed embedded system in a modern car [Car04]

thesis.

Giving the wide range of applications in which embedded systems are used, the requirements for embedded systems vary in terms of performance, size, cost, dependability etc. Many safety-critical applications impose physical, modularity and, of course, safety constraints. Following these constraints, it is not unusual that the implementations are distributed, consisting of different hardware components (nodes) interconnected in a network. An example of a distributed system as it can be found in a modern car is depicted in Figure 1.1.

Initially, each function was implemented in a separate hardware component (node) - the so-called *federated architecture*. In an airplane or in a car, the number of such nodes could reach over one hundred. An obvious disadvantage of the federated architectures is that they require a lot of hardware - (at least) one node for each function. Often, the hardware is replicated for achieving fault tolerance. Besides the pure hardware cost, there are other factors to be considered: space required by the hardware, power consumption, weight, wiring, maintenance [Rus99].

A solution was needed for reducing the number of nodes and, implicitly, the costs. The solution was provided by the so-called *integrated architecture* approach, where several functions are integrated into one node. Moreover, the current trends are towards the integration of functions of different criticality levels, as well as non-critical functions, onto the same node. Thus, we can now speak about *mixed-criticality* embedded systems, systems where applications with different levels of criticality share the same computing platform.

The concept of common resource sharing is applied in integrated architectures with the restriction that tasks having different criticality levels need to be separated (so they can't influence each other). Otherwise, for example, a faulty low criticality task could corrupt the memory of a high criticality task.

Avionics is a real-world example of a domain where this shift from federated architectures to integrated architectures has happened. There, the separation required between functions with different criticality levels is achieved at platform level with IMA (Integrated Modular Avionics, specified in the ARINC 651 standard) [Ari91] and at communication level with AFDX (Avionics Full Duplex Switched Ethernet, standardized as ARINC 664) [ARI09].

IMA specifies spatial and temporal operating system-level partitioning mechanisms [Ari97] that make possible the execution of mixed-criticality application on the same processing node. AFDX achieves the required separation at network level through communication multiplexing. Other communication-level separation solutions are available such as SAFEBus [HD93], TTP [Kop11] and the TTEthernet protocol [as611]. The TTEthernet protocol is based on Ethernet and is AFDX compliant. In this thesis, TTEthernet is the technology considered for communication.

In avionics, as well as in other safety-critical systems, it is essential that the data is delivered from the sender to the receiver in a timely fashion, up-to-date and complete. Otherwise said, deterministic guarantees are needed for the end-to-end communication delays. AFDX provides mechanisms for guaranteeing the determinism of the avionics communications.

In Chapter 2, we will give an overview of two of the state-of-the-art technologies used to implement today's integrated architectures. The first part of the chapter deals with IMA ("Integrated Modular Avionics"), which provides the partitioning mechanisms necessary at platform level for the integration of mixed-criticality applications. The chapter's second part is dedicated to the presentation of the TTEthernet protocol, which provides support for applications of mixed-criticality at network level. In Chapter 3 we propose a response time analysis for tasks scheduled on IMA-like partitioned architectures. In Chapter 4 we present the existing TTEthernet timing analysis, as well as the state-of-the-

art timing analyses for AFDX whose applicability to TTEthernet is discussed. In the last part of Chapter 4 we describe the TTEthernet simulator that we have implemented and evaluated against the existing TTEthernet timing analysis. Finally, we present our conclusions and some directions for future work in Chapter 5.

1.1 Thesis Objectives

The objective of the thesis is to develop and evaluate analysis methods for mixed-criticality applications implemented on distributed architectures that use TTEthernet as communication technology. So we are concerned both with the analysis of mixed-criticality tasks running on processing units and with the analysis of mixed-criticality messages sent over the network.

At the CPU-level we consider IMA to be used as the partitioning solution. This means that applications with different criticality levels will run on the same computing node as long as they are run in separate partitions. Each partition can have its own scheduling policy and it is our assumption that the applications are scheduled using either a time-triggered approach (static cyclic scheduling (SCS), for example), or an event-triggered approach (like fixed-priority preemptive scheduling (FPS)).

At the communication level we use the TTEthernet protocol. TTEthernet uses the concepts of "virtual links" to provide spatial separation between messages of different safety-criticality. Regarding time-criticality, TTEthernet provides three traffic classes: Time-Triggered (TT), Rate Constrained (RC) and Best Effort (BE), which support the transmission of messages with different time constraints.

We assume that the hardware platform is given (including the topology if the virtual links), and that the designer has specified the mapping of the tasks to processing elements and of messages to virtual links. Furthermore, we consider as input the assignment of tasks to partitions (which can implement both SCS or FPS scheduling) and of each message to one of the two time-critical traffic classes (TT or RC).

In such a context, the thesis objectives are:

- Propose a new schedulability analysis for FPS tasks such that the partitions are taken into account. The current state-of-the-art schedulability

analyses cannot handle partitions, or are too pessimistic when taking partitions into account.

- Implement the schedulability analysis in a tool and evaluate its pessimism by comparing it to the existing analyses on several synthetic and realistic benchmarks.
- Propose a new schedulability analysis for RC messages on TTEthernet networks, which can take into account the TT message schedules. There are many approaches to the analysis of RC messages on AFDX. However, there is a single proposed analysis for RC messages on TTEthernet, which makes very restrictive assumptions and, thus, is overly-pessimistic.
- Implement the analysis of RC messages in a tool and evaluate it by comparing it to the existing analysis on several benchmarks. Due to time constraints, and the inherently difficulty of the TTEthernet analysis, we were not able to finish the implementation of our proposed analysis. Instead, we have implemented a TTEthernet simulator, which has been used to evaluate the pessimism of the existing RC message analysis. In addition, we have also corrected and extended the Java implementation of the currently existing RC message analysis.

The methods and tools have been extensively described, tested and evaluated.

Chapter 2

Partitioned Architectures

As we indicated in the previous chapter, the subject of this thesis is the analysis of mixed-criticality application implemented on TTEthernet-based distributed architectures. In this chapter we give an overview of the concepts and technologies underlying this type of applications.

In the first part of the chapter we will present the IMA design concept for mixed-criticality applications. In the second part of the chapter we will focus on the TTEthernet protocol. We will present its main characteristics and we will also give a more detailed account of its inner workings through an example.

2.1 Integrated Modular Avionics

We can distinguish two approaches largely employed in avionics concerning the architecture styles in which the digital flight control functions are implemented. One approach is to use a federated architecture, the other one relies on a so-called integrated architecture.

In a federated architecture each flight function (like autopilot, yaw damping etc.) is implemented on its own computer system (or, using another term from the

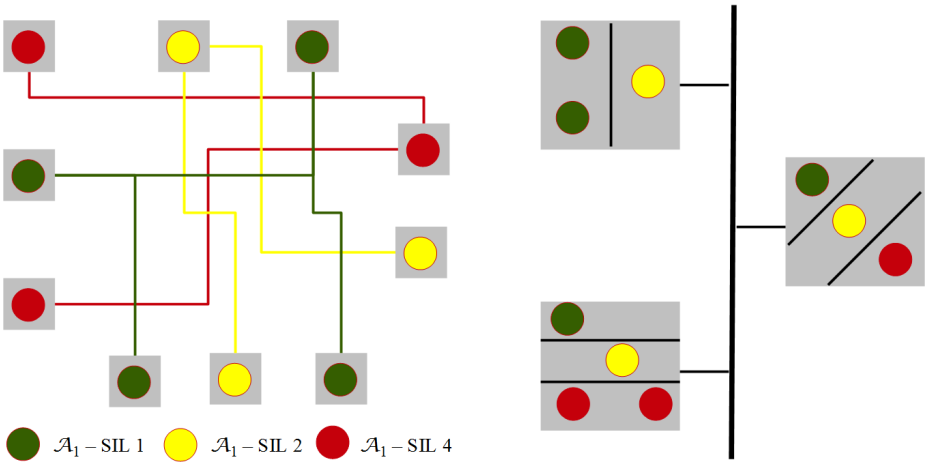
literature, node). The computer system of one function is only loosely coupled to the computer systems of other functions [Rus99]. In such an architecture, fault containment is intrinsically achieved. Because of the loosely coupling of the computer systems performing different functions, a (software or hardware) fault in the computer system performing a certain function can not propagate to the other functions. Put differently, in a federated architecture the components interact with one another (where it is necessary), so there is information exchange between different components, but the components can be designed in such a way that they're immune to faulty data coming from other components.

An obvious disadvantage of the federated architectures is that they require a lot of hardware - (at least) one node for each function. Often, the hardware is replicated for achieving fault tolerance. Besides the pure hardware cost, there are other factors to be considered: space required by the hardware, power consumption, weight, cooling, installation, maintenance [Rus99]. Moreover, the complexity of the avionics systems was doubling every five years [CC93], limiting the evolution possibilities of the federated architectures.

The alternative to the federated architecture style is the IMA (Integrated Modular Avionics) architecture, considered to be today's state-of-the-art in avionics. IMA is used, for example, in Airbus A-380, F-22, F-35 and Boeing 787 [Ram07]. Although, there is no overall standard for this integrated architecture, IMA is supported in practice by standards such as ARlNC 651 ("Design Guidance for Integrated Modular Avionics") [Ari91], ARINC 653 ("Avionics Application Software Standard Interface", where the APEX API is defined) [Ari97], ARINC 659 ("Backplane data bus") [Ari93] and ARINC 629 ("Multi-Transmitter Data Bus; Part 1, Technical Description (with five supplements); Part 2, Application Guide (with one supplement)") [Ari96].

In IMA we have multiple functions sharing the same computing platform. Several functions can be integrated onto the same node and the nodes are interconnected in a network. Because of the existence of this shared computing platform in IMA, the fault containment between functions is not as strong as in the federated architecture. For example, one function can monopolize the computing resources (memory, processor, bus), affecting this way all the other functions sharing the same computing platform. It can be also possible for one function to corrupt the memory used by another function; or for the error in one function to propagate to and perturb other functions. The only way in which functions can be protected against this kind of scenarios is to have a partitioning mechanism. This partitioning mechanism should protect against fault propagation from one function to another to a degree equivalent to the one inherently existing in the federated architectures.

The advantage of integrated architectures is that, by integrating more functions



(a) Federated architecture **(b)** Integrated architecture

Figure 2.1: Evolution of avionics architectures

on one node, the number of nodes is reduced. This leads to a decrease in the pure cost of hardware, the wiring and space required, the power consumption, the maintenance cost etc..

An illustration of the differences between the two architectural paradigms we discussed is given in Figure 2.1. The three applications shown in the figure have different criticality levels. Their criticality levels are captured by their associated Safety Integrity Levels (SILs). There are four SIL levels - 1, 2, 3, 4 - with 4 being the most critical and 1 being the least critical. The gray squares that appear in the two sides of the figure represent the computing nodes on which the applications are executed. Figure 2.1a shows a possible distribution of the applications' functions in a federated architecture, while Figure 2.1b shows how the same functions could be distributed over an integrated architecture. In the case of the integrated architecture, due to their different criticality levels, the applications are placed in separate partitions (in our figure, the partitions on a processing node are separated by black lines).

One can notice that the advantage of intrinsic fault containment in the federated architectures also implies a cost in poorly coordinated control and fault-prone pilot interfaces [Rus99]. Actually, the allocation of flight automation to separate functions in the federated architecture is mostly the result of largely accidental historical factors [Rus99]. It can happen that control variables that are tightly coupled in a dynamical sense (for example, engine thrust and pitch angle) [Rus99] to be managed by separate functions. A change in either one of

these coupled variables implies a change in the other one and, since there is no coordinated control for them, simple services employing these variables (such as cruise speed control or altitude select) end up having too complex implementations that are difficult to manage [Rus99]. The aforementioned exaggerated complexity is revealed by the multiple modes and submodes employed in the functions in the federated architecture. This complexity creates problems in the flight crew's interaction with the automation systems and also increases the costs with development and certification. Without an integrated control, this complexity is unavoidable.

Another problem with the fact that the federated architecture uses many computer systems is that, typically, the computer systems implementing different functions are different. This diversity in platforms increases significantly the cost of the development and of the certification of the software that needs to run on these platforms. IMA brings the advantage of its hardware platform standardization.

Hardware fault tolerance in the federated architecture is achieved by replication. Critical flight functions run on replicated hardware (typically, quad-redundant or greater for primary flight control, triple for autopilot and auto-landing, dual for flight management) [Rus99]. In these conditions, the costs of achieving hardware fault tolerance in a federated architecture are significant (there can be even 50 processors needed), but the operational flexibility is limited: a single faulty processor in any function can keep the plane on the ground [Rus99]. On the other hand, IMA has the advantage that its processors are not tied to a specific function. The replicated processors can be allocated according to the necessities as long as the total number of non-faulty processors is sufficient to provide the level of replication needed for each function [Rus99].

2.1.1 Partitioning in IMA

We already said that the partitioning is a protection mechanism against propagation of faults from one function to other functions. Otherwise put, the purpose of partitioning is fault containment [Rus99]. It is important to clarify which is the scope of this partitioning that we're discussing about. Is partitioning considered only for limiting fault propagation between computer systems (nodes) implementing different functions or it is considered also for limiting fault propagation within nodes? If there would be no internal partitioning in the nodes (that is, partitioning between different applications running on the same node and fulfilling different functions) and only partitioning between nodes would be employed, then one disadvantage clearly arises. The disadvantage is that all applications running on a certain node should be certified at the highest

criticality level amongst them. This would lead to significant increases in development and certification costs. With internal partitioning, the functions can be decomposed into software components with different criticality levels, this way reducing the development costs while localizing where the assurance effort should be put [Rus99]. In these conditions, the answer to our previous question is obvious: partitioning must be considered within nodes, not only between them.

In IMA, different functions share the same resources (such as computing units, buses, peripherals). This resource sharing facilitates fault propagation and the function of partitioning is to not allow for this fault propagation to happen. In [Rus99], a benchmark for the effectiveness of partitioning is introduced. The benchmark is called the "Gold Standard" and is presented in the following:

"A partitioned system should provide fault containment equivalent to an idealized system in which each partition is allocated an independent processor and associated peripherals and all inter-partition communications are carried on dedicated lines."

This partitioning requirements' formulation has the problem that it is hard to use in real life, since the software that will run in these partitions will not be developed and tested in the conditions described in the Gold Standard. The idealized system dedicated to a function alone that is mentioned in the standard is just an imaginary artifact, and can not really be used in the function's development, testing and certification. The only available environment is constituted by the partitioned system itself, so a reconsideration of the Gold Standard in terms of the available systems is necessary. The "Alternative Gold Standard for Partitioning" is such a reconsideration [Rus99]:

"The behavior and performance of software in one partition must be unaffected by the software in other partitions."

The standard says that the software in one partition must be unaffected by the software in other partitions. But it rarely happens that the software functions executing in different partitions are completely independent. On the contrary, frequently data and control inputs are exchanged between functions. This points out one of the hazards to be avoided: that a fault in one partition can corrupt the data, the control inputs or the code specific to another partition. Another type of hazard to be avoided appears more obvious if we read again the Gold standard. It refers to the ability of functions in one partition to gain access or service from a shared resource like a processor, peripherals or a bus [Rus99].

In regard of these hazard types, according to [Rus99], there are two classes of partitioning that can be devised: temporal and spatial. They will be presented

below.

Spatial partitioning:

"Spatial partitioning must ensure that software in one partition cannot change the software or private data of another partition (either in memory or in transit) nor command the private devices or actuators of other partitions." [Rus99]

The resources are statically allocated to partitions by the system integrator, so the system integrator needs to make sure that it allocates enough resources to each partition and that the spatial separation between partitions is respected.

One way to ensure that the spatial segregation in avionics is respected is through hardware mediation performed by a memory management unit (MMU). It is the responsibility of the operating system to protect the partition data from being modified by software running in other partitions. With hardware mediation this is achieved through different modes of operation for the processor. The idea is to have at least two operation modes (user and supervisor) and to verify with the MMU all the memory accesses done in user mode by the processor. The MMU contains tables with the addresses that can be accessed by each partition and the operating system kernel makes sure that these addresses don't overlap.

Since the spatial partitioning forbids shared data areas between partitions, the applications in different partitions can exchange data only through inter-partition communication. The inter-partition communication is done through messages sent over the AFDX network in case the partitions are not on the same module and through blackboards, buffers, event or semaphore services in case they are.

Temporal partitioning:

"Temporal partitioning must ensure that the service received from shared resources by the software in one partition cannot be affected by the software in another partition. This includes the performance of the resource concerned, as well as the rate, latency, jitter, and duration of scheduled access to it." [Rus99]

The parameters that temporally define a partition and the partitions' scheduling are specified offline by the system integrator. The temporal attributes of a partition are: duration (which needs to be calculated considering the amount of time required by the functions executing in the partition), deadline and period. Partitions have no priority. The operating system ensures that the scheduling and the temporal allocations for the partitions are respected. Every partition must be allocated execution time equal to its duration, but not mandatory at once, so the duration can be split over several execution windows. Once the

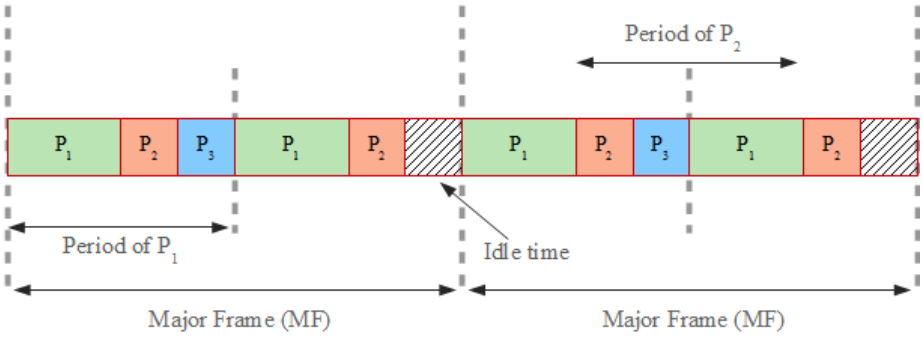


Figure 2.2: Time partitioning in IMA

temporal execution window allocated to a partition ends, the execution of the partition's functions is suspended and the operating system starts the execution of the next partition in the schedule. Partitions in an avionics module are grouped into a Major Frame (MF), as it can be seen in Figure 2.2. The major frame has a fixed duration and is repeated continuously during the module's functioning. Its period (which is equal to its duration) defines the periodicity of the scheduling and is calculated as the least common multiple of the periods of the partitions composing the major frame. A major frame can also contain idle time, which is not allocated to any partition.

In conclusion, we repeat that the purpose of partitioning is fault containment and for this it must block both the spatial and temporal pathways for fault propagation [Rus99].

Although the concept of partitioning was discussed in the context of avionics, it can be easily seen that it is not exclusive to this field. It is a general concept with a large area of application in embedded systems, from avionics and medical devices to automotive, wherever mixed criticality functions are to be run on the same platform.

2.2 TTEthernet

In safety-critical areas (like the automotive or aeronautic industries), the time-triggered communication paradigm is widely employed because of the deterministic behavior it offers. One condition for employing the time-triggered approach in a distributed environment is to assure that the systems in the network are synchronized (that is, they have the same perception of the current moment in

time). Once this condition is fulfilled, the systems in the network just need to follow a static communication schedule. This communication schedule is typically built and verified offline so that no resource conflicts on the network can yield surprises.

However, besides the time-triggered approach in the scheduling of messages and tasks, there exists another complementary approach - the event-triggered paradigm. These two approaches have been long analyzed [ATB93, Kop11] in terms of flexibility, jitter control, predictability etc. Each approach has advantages and disadvantages and the consensus in the real-time and embedded systems community is that the best approach to use depends on the application [TSP12], this also implying that both approaches can be simultaneously employed in a system.

The time-triggered communication paradigm is the appropriate approach for applications with high criticality temporal and fault tolerance requirements [Ste11]. Typically, in the mixed-criticality systems, the applications with high criticality temporal requirements will coexist and share the physical network with less demanding applications [Ste11]. The integration of these applications with different criticality levels should optimally use the available resource, but it was shown in [Foh94] that, more likely, the use of static schedules will eventuate in resources not being optimally used (i.e., idle times in between tasks) [Ste11]. A similar observation was made for statically scheduled networks and led to the development of the TTEthernet design [SBH+09, Ste11, KAGS05].

TTEthernet can handle both event-triggered and time-triggered traffic. One of the design rationales behind TTEthernet was to obtain a "novel unified communication architecture based on Ethernet that meets the requirements of all types of non-real-time, real-time, and multimedia applications up to the most demanding safety-critical real-time products, while still maintaining full compatibility with the existing Ethernet standard." [KAGS05].

TTEthernet is originating from the academic TT-Ethernet technology [KAGS05], which on its turn is originating from the standard Ethernet and TTP [Kop11]. Currently, TTEthernet is developed by TTTech ComputerTechnik AG and is standardized in SAE AS6802 [as611].

It is worth mentioning that TTEthernet is compliant with the AFDX (or ARINC 664 Specification Part 7) standard [ARI09]. As mentioned in section 2.2.1, the communication provided by AFDX is event-triggered and deterministic. TTEthernet is supporting event-triggered communication, using for this the free bandwidth available after the static communication schedules for the time-triggered communication were determined. Since the AFDX concepts regarding the event-triggered traffic are relevant also for TTEthernet, we will briefly present them

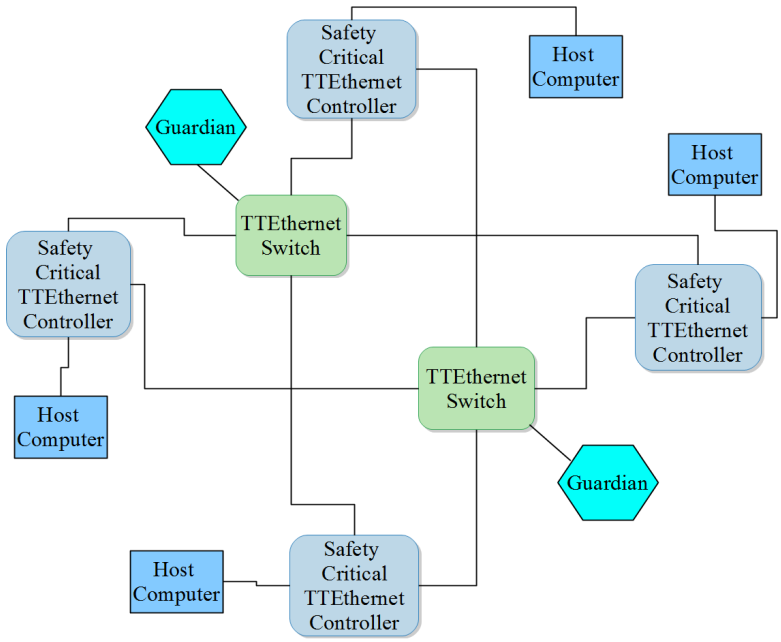


Figure 2.3: A safety-critical TTEthernet configuration

in section 2.2.1.

By adding determinism to the classical Ethernet, TTEthernet qualifies as a communication technology for mixed-criticality applications. The required separation between different criticality levels is achieved, at temporal level, through the static TT communication schedules and the bandwidth allocation for RC messages. The spatial separation is achieved through the concept of virtual links.

Time-triggered communication requires a synchronized time base across the network to be maintained. TTEthernet uses synchronization messages (the so-called protocol control frames) to maintain a global notion of time. TTEthernet traffic can coexist on the same physical network with Ethernet or AFDX traffic (that is, TTEthernet is a transparent synchronization protocol [as611]). In these conditions, TTEthernet is able to maintain the dispatch order and the relative timing of the TTEthernet messages based on its transparent clock mechanism [as611].

Using TTEthernet, depending on the specifics of the supported applications, one can build safety-critical configurations, as well as standard configurations

(appropriate for non-critical applications). TTEthernet services are built with emphasis on modularity, so it is also possible to construct configurations for intermediate safety requirements. In Figure 2.3 we present an example of a safety-critical TTEthernet configuration. The TTEthernet switches used in safety-critical configurations are the same as the ones used in standard configurations, but in the safety-critical configurations the switches are monitored by guardians. The guardians constitute an independent fault-containment unit, that monitors the operation of the switch and of the controllers connected to the switch [KAGS05]. In case it detects errors, the guardian can disable the inputs and outputs of the TTEthernet switch.

Compared to the safety-critical configuration presented in Figure 2.3, a standard configuration would not employ guardians and, instead of safety-critical TTEthernet controllers, it would employ standard TTEthernet controllers. Note that in a standard configuration, due to the TTEthernet's compatibility with Ethernet, standard Ethernet controllers are also accepted. There is no restriction on the number of controllers in the network.

2.2.1 Avionics Full Duplex Switched Ethernet

AFDX (implementation of the ARINC 664 Part 7 standard [ARI09]) is today the reference communication technology in avionics. The explosion of the number of nodes in the federated architectures challenged the communication infrastructure in terms of bandwidth, weight and number of needed buses. AFDX answered to these challenges by multiplexing communication flows over a full duplex switched Ethernet network. Because the network links are full duplex, the contention problem is eliminated. Being based on Ethernet, the data rates provided by AFDX are much higher than the ones provided by its predecessor ARINC 429.

An AFDX network is composed of end systems and switches connected trough full duplex links. Through the end systems, the avionics subsystems send to and receive data from the network. Besides connecting the avionics subsystems to the switches, the end systems also perform traffic shaping. The switches perform traffic policing and routing of the data packets to their corresponding end systems receivers. Each end system is connected to a single switch port and, correspondingly, each switch port is connected to a single end system. An illustrative AFDX network is depicted in Figure 2.4. We can distinguish four switches (SW_1 to SW_5) and seven end systems (ES_1 to ES_7). We notice that the previously mentioned restrictions regarding the interconnection of network elements are respected.

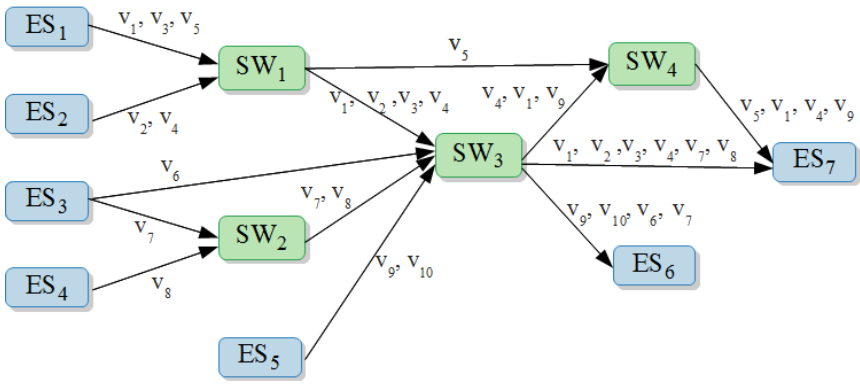


Figure 2.4: An example AFDX network.

The fundamental concept in a AFDX network is the virtual link (VL). A virtual link (VL) is defined in [ARI09] as a "conceptual communication object", a virtual communication channel [BSF09] through which the end systems exchange messages. A virtual link has the following properties:

- it defines a "logical unidirectional connection from one source end-system to one or more destination end systems" [ARI09]. That is, only one end system can be the source of an virtual link. A switch can not be the source or the destination of a virtual link, it can be only a part of a VL path. For routing packets, the AFDX switches use virtual link IDs.
- for each virtual link, a maximum bandwidth was designated off-line, by the system integrator. In this regard, the following VL properties are important: the BAG (Bandwidth Allocation Gap) and the minimum and the maximum frame length (s_{min} and s_{max}).

The BAG is the minimum time interval between two consecutive frames belonging to the same virtual link. Therefore the maximum usable bandwidth for a VL can be calculated as s_{max}/BAG.

In Figure 2.4 there are ten virtual links, some of which are unicast (only one receiver), the rest being multicast (multiple receivers). For example, v_5 is an unicast VL with the path (ES_1 - SW_1 - SW_4 - ES_7) and v_9 is a multicast VL with the paths (ES_5 - SW_3 - SW_4 - ES_7) and (ES_5 - SW_3 - ES_6).

All the virtual links in an AFDX network (that is, their routing and their bandwidth parameters) are statically defined. This is important because in the safety-

critical area of avionics deterministic timing guarantees are required and having the virtual links' parameters statically defined allows the offline calculation of the maximum end-to-end network delays.

AFDX needs to assure deterministic avionics communications. Analyzing the AFDX network for a given flow, we observe that the transmission delays in the network links can be easily computed since they depend only on the transmission rate and on the frame length. What are not so easy to evaluate are the latencies in switches experienced by a flow. The flows in an AFDX network are asynchronous and they can enter in competition over a switch's output ports. In this context, it is necessary to analyze for each flow the latencies in the switches' output ports, so that the upper bounds on the network end-to-end delays can be determined. We will talk in more detail about the state-of-the-art methods for determining worst-case end-to-end delays in AFDX networks in section 4.2.

2.2.2 Traffic Classes

We reiterate that TTEthernet supports both event-triggered and time-triggered communication, distinguishing for this purpose two fundamental traffic classes: standard Ethernet (ET) traffic and TT Ethernet traffic. These can be further divided in the following three classes of traffic: time-triggered (TT) traffic, rate-constrained (RC) traffic and best-effort (BE) traffic.

The time-triggered (TT) traffic is suitable for applications that require tight latency, predictability and jitter requirements. The TT traffic has the highest priority and it is driven by static communication schedules. The communication schedules are locally stored in the network's senders and switches. For the time-triggered communication to work, the systems participating in the conversation need to be synchronized. Synchronized time is essential for the isolation of critical applications from the less critical ones and for partitioning [as611].

The rate-constrained (RC) traffic is respecting the RC communication paradigm specified by AFDX. The idea behind RC communication is to guarantee that two consecutive frames belonging to the same RC flow are never sent back-to-back, but are always offset by a minimum pre-configured duration [SBH+09]. The inter-frame gap is assured for each RC flow on the end systems side by a traffic shaping function. A faulty end system may attempt to send frames more often then it should. In this scenario, the network is protected by the switches that implement a traffic policing function (the leaky bucket algorithm). This function checks if the message flow generated by an end system is indeed well-shaped. In case the minimum interval between two successive frames is not respected, the switch drops the too-early frames. In the RC traffic there is no synchronized

time base, the messages being asynchronous. This implies that multiple senders can simultaneously send frames. These RC frames may end up accumulating in the network switches and, consequently, have increased transmission jitter. Also, for making sure that switches' buffers have enough space so no message is lost or for calculating the buffers' necessary capacity, peak-load scenarios need to be considered at design time. Since the network parameters (buffer sizes etc.) and the frames' transmission rates are apriori known, the frames' jitters and latencies can be calculated offline. That's why we can say that the "RC transfers guarantee sufficient bandwidth allocation for each transmission, with defined limits for delays and temporal deviations" [as611]. Therefore, the RC traffic has bounded end-to-end latencies being suitable for applications which have less strict determinism and timing requirements.

The third traffic class supported by TTEthernet is Best Effort (BE). The BE traffic is analog to the normal Ethernet traffic; it doesn't offer any temporal or deterministic guarantees. This means that there is no guarantee that a message will reach its receiver and, in case it will, there is no guarantee regarding the message's delay. The BE traffic has the lowest priority among the TTEthernet traffic classes and it utilizes the bandwidth left unused by TT and RC transfers. Taking into account the lack of quality of service guarantees, the BE traffic class is not suitable for safety-critical communication.

2.2.3 Dataflow Integration

TTEthernet needs to handle the contentions that will most probable occur on the end systems' and switches' output ports when dataflows of different traffic classes are integrated onto a single physical network. In case contention occurs between messages of the same traffic class, the messages will be serviced according to the FIFO policy [SBH+09]. The situation in which a low priority message becomes ready while a high priority message is transmitted has a straightforward solution: the transmission of the high priority message is continued and the low priority message is queued.

The handling for the case in which a high priority message becomes ready while a low priority message is transmitting is more complex, TTEthernet providing three integration methods for this situation. Let's discuss them considering the example TTEthernet cluster in Figure 2.5 composed of three end systems (ES_1, ES_2, ES_3) and one network switch (SW_1). A possible Gantt chart with the traffic on the physical links in the system is shown in Figure 2.6. The two messages in the system (the TT message tt_1 and the RC message rc_1) are represented by colored rectangles. The left edge of the rectangle marks the start of the message's transmission on the physical link. Correspondingly, the right

edge of the rectangle indicates the ending of the transmission. tt_1 is sent by ES_1 to ES_3 through SW_1, while rc_1 has the same destination, but a different origin (ES_2).

We can see that the high priority TT message tt_1 reaches the switch SW_1 after the lower priority RC message rc_1. The three TTEthernet integration methods ((a) preemption, (b) timely block and (c) shuffling) are illustrated in Figure 2.6 by the three possible scenarios for the traffic on the $SW_1 - ES_3$ link.

When the integration policy employed is preemption, the transmission of the low priority message (rc_1 in our case) is stopped and (after establishing the minimum amount of silence on the channel) the high priority message (tt_1) is relayed. The transmission of the low priority message is restarted after the high priority message finished transmitting. Stopping the transmission of rc_1 implies that ES_3 will receive a truncated version of this message. TTEthernet provides mechanisms for avoiding that the receiver of a truncated message interprets it as correct. When rc_1 is transmitted again, after tt_1, it is transmitted integrally. One drawback of this method would be then that each truncation leads to a loss of bandwidth [SBH+09]. One advantage is that, in regard to high priority messages, the switch guarantees a known and constant latency.

The timely block integration method implies that the switch will not forward RC messages when TT messages are expected [SBH+09]. Since the TT messages are transmitted based on static schedules, the switch can appreciate if the transmission of a low priority message will finish before a TT message needs to be relayed. This can be seen in Figure 2.6, in scenario (b), where rc_1 is blocked (postponed) from transmission until tt_1 frees the communication link. The amount of time while rc_1 is blocked and the switch doesn't transmit anything is represented by a hatched rectangle. In case the length of the lower priority message is not known, the switch will have to consider the maximum possible length of the lower priority message when calculating the timely block interval. The timely block approach guarantees that the outgoing ports will be available for the high priority messages, so the delays due to integration are known and constant for these messages.

In the case of the third integration method (shuffling), the arrival of a high priority message doesn't interrupt the ongoing transmission of the low priority message. So, the high priority message is delayed until the low priority message is transmitted. This situation can be seen in scenario (c) of Figure 2.6. In this case, the maximum delay that can be experienced by a high priority message is equal to the maximum transmission duration of a low priority message. From a utilization point of view, shuffling is an optimal solution [SBH+09], since the outgoing ports will not be blocked for low priority messages and these message will not be truncated. However, with shuffling, the real-time quality of the

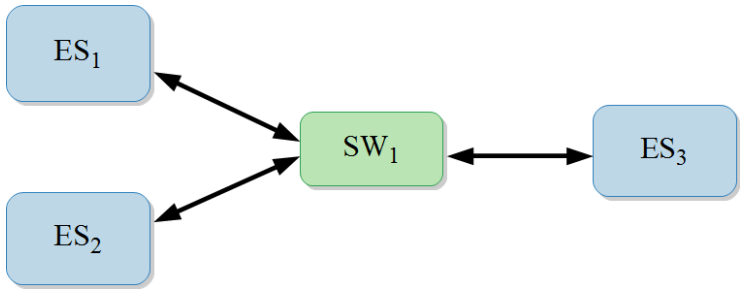


Figure 2.5: TTEthernet cluster

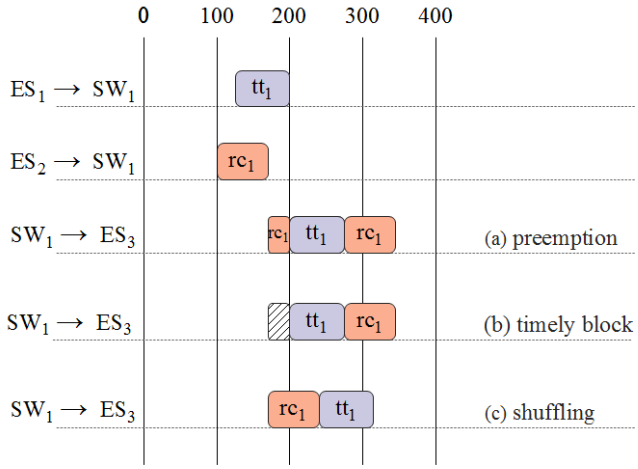


Figure 2.6: Integration policies for RC and TT traffic

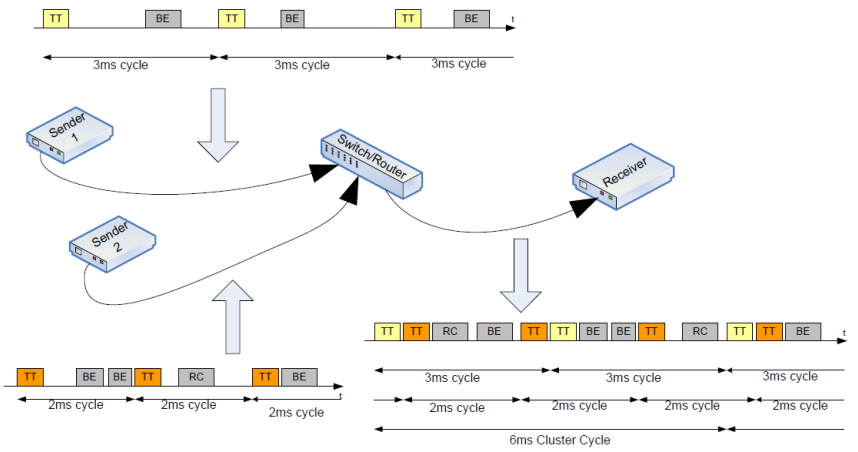


Figure 2.7: Integrated traffic in a TTEthernet network [as611]

TT traffic suffers in terms of transmission jitter and latency. Since the TT traffic is relaying on a synchronized time base, the dispatch points in time for the TT messages are known apriori to the receivers. The transmission jitter can be mitigated by including in the messages of low priority traffic class a global timestamp informing the receivers of their dispatch time [SBH+09]. The increased latency and jitter imposed by the shuffling policy affects the static communication schedule for the TT traffic. For this reason, the changes of the TT timing which can appear due to the RC and BE traffic need to be considered when creating the communication schedule. It is considered that, in 100 Mbps or 1 Gbps networks, shuffling provides good-enough real-time quality for a big variety of applications in automotive and avionics [SBH+09].

A concrete example of possible traffic in a TTEthernet network composed of two senders, one receiver and one switch is presented in Figure 2.7 [as611]. One sender is transmitting a TT frame every 3 ms and also some BE frames. The other sender emits some BE frames and TT frames with a period of 2 ms. The switch gets the frames from the sender end systems and composes the integrated dataflow that reaches the receiver.

2.2.4 Basic TTEthernet Modeling Concepts

In what is next we present a model for the TTEthernet networks as it is proposed in [Ste11, TSP12, Ste10b]. A TTEthernet network can be formally defined as

an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. The set of vertices $\mathcal{V}$ is the set of the end systems and switches in the network. The set of edges $\mathcal{E}$ is the set of the physical links that connect the vertices. In the TTEthernet cluster from Figure 2.8, $\mathcal{V} = \{ES_1, ES_2, ES_3, SW_1, SW_2\}$, and the physical links composing set $\mathcal{E}$ are represented with double black arrows.

The physical links are modelled as dataflow links. A dataflow link connects one sender vertex to one receiver vertex. Since the physical links are full duplex (they allow communication in both directions), a physical link defines two dataflow links. More formally, the dataflow link $l_i = [v_x, v_y] \in \mathcal{L}$, where $\mathcal{L}$ is the set of all dataflow links in the network, represents the direct communication channel between v_x and v_y. In Figure 2.8, examples of dataflow links are l_1 and l_2 connecting the switch SW_1 with the end systems ES_1 and ES_2. An ordered sequence of dataflow links connecting one sender to one receiver forms a dataflow path. A dataflow path denoted dp_i, connecting a sender v_1 to a receiver v_x, can then be expressed as $[[v_1, v_2], ..., [v_{(x-1)}, v_x]$. The dataflow path dp_2 depicted in Figure 2.8 can be then expressed as $[[ES_3, SW_2], [SW_2, SW_1], [SW_1, ES_2]]$. The set of the dataflow paths in the network is denoted by $\mathcal{DP}$.

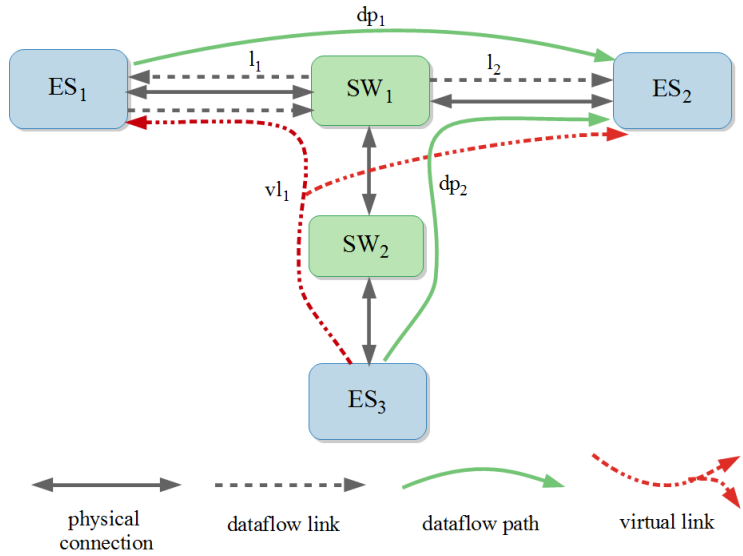


Figure 2.8: TTEthernet network cluster example

A dataflow path represents a unidirectional connection from a sender to a receiver. Virtual links, as we explain in section 2.2.1, represent logical unidirectional connections between one sender end system and multiple receiver end systems. A virtual link can also be thought of as a directed tree, the sender

being the root and the receivers being the leafs. The virtual link vl_1, illustrated with red dot-dash arrows in Figure 2.8, is a tree with root ES_3 and leafs ES_1 and ES_2. In this context, a virtual link vl can be expressed as the reunion of the dataflow paths connecting the sender to each of the receivers: $vl = \bigcup dp_i$. The set of all virtual links in the system is denoted by $\mathcal{VL}$. Virtual links are a way to achieve spatial partitioning between messages with different criticality levels transmitted over the network.

Messages that are transmitted over the network are packed into frames. We assume that each message is packed into exactly one frame and each frame carries only one message. The issue of frame packing, addressed for example in [PEP05], is orthogonal to our problem. We denote the set of all frames in the network by $\mathcal{F}$, where can be further divided into $\mathcal{F}^{TT}$, $\mathcal{F}^{RC}$ and $\mathcal{F}^{BE}$ which represent the sets of frames belonging to the three traffic classes provided by TTEthernet ($\mathcal{F} = \mathcal{F}^{TT} \cup \mathcal{F}^{RC} \cup \mathcal{F}^{BE}$). The identification of a frame's traffic class is done based on a bit pattern in the frame's header. Although in a multi-clustered TTEthernet network, a frame can belong to different traffic classes in different synchronization domains, we assume in our model that the frames keep in the whole network the traffic class they have been assigned by the designer. An equivalent assumption would be to consider our model only for one TTEthernet cluster or to consider the network as composed by one cluster. The instance of a frame $f_i \in \mathcal{F}$ on a dataflow link $[v_x, v_y]$ is denoted by $f_i^{[v_x,v_y]}$.

A TT frame $f_i^{[v_x,v_y]}$ is completely temporally specified by the following set of parameters: $\{f_i.period, f_i^{[v_x,v_y]}.offset, f_i.size, f_i.deadline\}$. A RC frame $f_i^{[v_x,v_y]}$ is fully described by the parameters $\{f_i.rate, f_i.size, f_i.deadline\}$. The TT frames are periodic, while the RC frames are not necessarily periodic, the RC frames having a minimum inter-arrival time. In this respect the rate of the RC frames can be seen as $f_i.rate = 1/f_i.period$. The offset $f_i.offset$ of a TT frame f_i represents the sending time of a frame relative to its period. The size, the deadline, the period (of the TT) or the rate (of the RC) frames are configured offline by the designer, so in our analyses these parameters are given apriori. Based on the size of the frame f_i and on the speed of the physical link $[v_x, v_y]$ it is easy to determine the transmission duration (denoted $C_i^{[v_x,v_y]}$) of the frame on the dataflow link.

2.2.5 How TTEthernet Works

This section will illustrate how TTEthernet works through the example in Figure 2.9 (taken from [TSP12]). We consider two applications, $\mathcal{A}_1$ (composed of tasks τ_1 and τ_3) and $\mathcal{A}_2$ (consisting of τ_2 and τ_4). Because they are of differ-

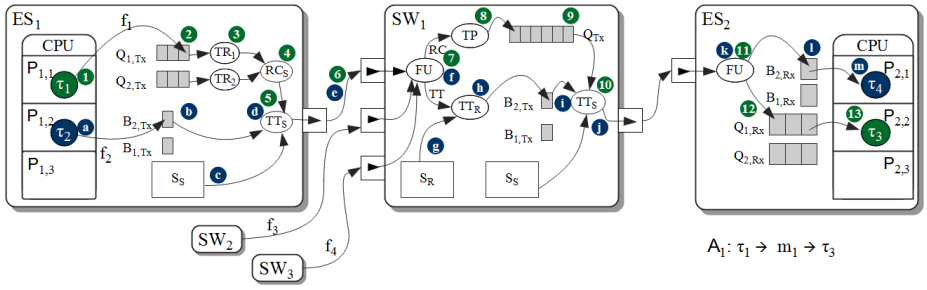


Figure 2.9: Example transmission of TT and RC messages [TSP12]

ent criticality levels, the applications are executed in separate partitions on the processing elements. The separation at the communication level between the applications' messages is achieved by sending the messages on different virtual links. The TTEthernet network supporting the communication in our example consists of two end systems (ES_1 and ES_2) and three switches (SW_1, SW_2 and SW_3).

We consider that task τ_1 mapped on ES_1 sends the RC message m_1 to task τ_3 on ES_2 and that task τ_2 mapped as well on ES_1 sends the TT message m_2 to task τ_4 on ES_2 [TSP12]. The messages are considered to fit in one frame each (m_1 is packed in frame f_1 and m_2 is packed in frame f_2). Like we previously stated, the frames are assigned to different virtual links, but the virtual links are not depicted in Figure 2.9, instead, what is depicted, is their trajectory with details of the transmission steps. We can see in Figure 2.9, how the frames f_1 and f_2 cross the switch SW_1 (which is also crossed by frames f_3 and f_4 coming from SW_2 and SW_3) in their way from ES_1 to ES_2. The transmission steps for the frames under analysis (f_1 and f_2) are marked in the figure with letters from (a) to (m) or numbers from (1) to (13) on blue, respectively green, background.

The details of the time-triggered transmission are illustrated using the TT message m_2 packed in frame f_2 and sent by task τ_2 on ES_1 to task τ_4 on ES_2. The transmission starts with the packing of the message m_2 into the frame f_2 - step (a). In the next step, (b), the frame is put in the transmission buffer $B_{2,Tx}$ from where, according to the static send schedule kept in ES_1 - (c) -, it is taken and transmitted to the switch SW_1 by the scheduler task TT_S - (d). Conceptually, for each TT message sent from an end system there is a transmission buffer like $B_{1,Tx}$ [TSP12]. We reiterate that all the end systems and switches in the network store the static communication schedules based on which the TT communication is performed. There are both receive and send schedules, denoted by S_R, respectively S_S, and these schedules are determined offline. So, the scheduler task TT_S will send frame f_2 to switch SW_1 at the time

specified in the send schedule S_S of ES_1. The TT sender task TT_S provides a fault-tolerance service against the so-called "babbling idiot" failure. This failure could manifest if, for example, a task would become faulty and start to send more TT messages than it should according to the schedule. In such a case, the TT sender task TT_S will transmit messages only as it is specified in the static schedule S_S and, thus, protect the network. The transmission of frame f_2, on a dataflow link, to switch SW_1 is marked in the figure as step (e). Upon arriving at the switch, the frame is checked for validity and integrity by the Filtering Unit (FU) hardware task - (f). This verification is done for all frames received by the switch, since it is essential for the overall network robustness that only valid frames are forwarded [ARI09]. If frame f_2 is found valid, then the frame is forwarded to the TT receiver task TT_R in step (h).

The TTEthernet switch provides mechanisms to ensure fault containment over the network. One of these services is implemented by the TT receiver task TT_R. Analogously to the TT sender task TT_S, the TT receiver task TT_R relies on the receive schedule S_R (marked with (g) on the figure) when handling the reception of a TT frame. More precisely, the TT_R task checks if the TT frames arrived according to the S_R schedule within a specified receiving window [TSP12]. The size of the receiving window is calculated offline based on the sending times in the send schedules, the integration policy employed in the network and the clock synchronization mechanism's precision [TSP12]. The TT frames that arrive outside this receiving window are considered faulty and are, consequently, dropped.

Continuing with our example, if frame f_2 arrived within the receiving window, then the TT_R task will place it in the transmission buffer $B_{2,Tx}$ - (h). From here, similar to the transmission process in the end system ES_1, the frame will be taken over by the TT sender task TT_S - (i) - when the time is right (that is, at the time specified for the frame in the S_S send schedule stored in the switch) and transmitted to the end system ES_2 - (j). The arrival of the frame at the end system and the checking of the frame's validity by the FU task in ES_2 is marked in the figure as step (k). If the validation is successful, the FU task will place f_2 in its dedicated receive buffer B_{2,R_x} - (l). The frame will be taken from the receive buffer by task τ_4 when the task will be activated - (m).

For illustrating the internals of the RC transmission, we will use the message m_1 packed in frame f_1 and sent by task τ_1 on ES_1 to task τ_3 on ES_2. The message m_1 will cross the switch SW_1 on its way. Like in the case of TT messages, the transmission of m_1 starts with its packing into frame f_1 - step marked as (1) on the figure. After packing the message, task τ_1 places it in the queue $Q_{1,Tx}$ - (2). Such a queue is assigned for every RC virtual link and has a role in the traffic control at the end system.

It is required from end systems to ensure that each virtual link is serviced according to its BAG, irrespective of the attempted use of bandwidth by other virtual links [ARI09]. We remind the reader that each virtual link is characterized by a so-called BAG (Bandwidth Allocation Gap) which specifies the minimum time interval between two consecutive frames belonging to the same virtual link. That is, the BAG (together with the maximum frame size) determines the maximum usable bandwidth for a virtual link. The control of the amount of bandwidth given to a virtual link is performed by the Traffic Regulator (TR) task according to the virtual link's BAG. The traffic is regulated on a per virtual link basis so that, for each virtual link, the end system will not send more than one frame in each BAG interval. Regardless of the instantaneous frame rate of the virtual links, the TR task will space the frames to respect the BAG [ARI09]. In our example, this regulation step done by task TR_1 is marked with (3).

In the case of an end system with multiple RC virtual links, the RC scheduler task RC_S has the role of multiplexing the RC flows coming from the traffic regulator tasks - (4). The consequence of this multiplexing is that the flows can be affected by jitter. Besides the interference from other RC frames, solved by multiplexing by the RC scheduler task, the RC frames need to endure the interference of the TT traffic, which has higher priority. The possible integration methods for the TT and RC traffic have been discussed in section 2.2.3. The TT sender task TT_S on ES_1, depending on the TT static schedules and on the integration policy used, will send frame f_1 to SW_1 at an appropriate time - (5). The transmission of f_1 on the $[ES_1, SW_1]$ dataflow link is marked with (6) on the figure.

Like the TT frames, at their arrival at end systems or switches, the RC frames are checked for validity and integrity by the Filtering Unit (FU) tasks - (7). Another fault-containment feature of the TTEthernet switches is implemented by the Traffic Policing (TP) task. The purpose of the TP tasks in switches is somehow analogue to the purpose of the TR tasks in end systems. The TP task checks if the bandwidth allocation for virtual links is respected at the receiving end - that is, checks that the time interval between two consecutive frame instances on the same virtual link is greater than or equal to the corresponding BAG. The TP task implements the so-called leaky-bucket algorithm, which drops the frames arriving at shorter intervals than the specified BAG. This way, a faulty end system sending RC frames at an erroneous rate can not disturb the network.

If f_1 passes the TP task's checks (8), then it will be placed in the outgoing queue Q_{Tx} - (9). The frames from the Q_{Tx} queue are typically handled in FIFO order by the TT sender task TT_S - (10). Like the TT sender task on ES_1, the TT sender task on SW_1 will determine the sending time of f_1 based on the

TT static schedules according to the TT and RC traffic integration policy used. After traversing the dataflow link $[SW_1, ES_2]$, f_1 will be checked for validity and integrity by the FU task on ES_2 - (11). If f_1 passes the verification, it will be placed in the receiving queue $Q_{1,Rx}$ - (12). The addressee task τ_3 can then read f_1 from the queue when it becomes active - (13).

Chapter 3

IMA Analysis

We saw in the previous chapter that the integration of mixed-criticality applications onto the same architecture is permitted only if there is enough temporal and spatial separation among them. This separation is achieved through partitioning and, in this chapter, we consider an IMA-like partitioning scheme.

In this chapter we will present our proposed schedulability analysis for tasks running in partitions that use fixed priority preemptive scheduling (FPS). The considered application and system models are presented in section 3.1 and, respectively, section 3.2. An example showing the importance of taking into account the partitions when performing the schedulability analysis is given in section 3.3. Our proposed response time analysis, based on the WCDOPS+ algorithm, is presented in section 3.4. The proposed analysis has been compared, in section 3.5, with the schedulability analysis for FPS tasks running in partitions proposed by Audsley and Wellings. Throughout this section we will denote the analysis proposed by Audsley and Wellings with SA (from "Schedulatibility Analysis"). Our analysis will be denoted by SA+.

Based on the work presented in this chapter, we wrote a paper [MTSAP12] which was presented in the work in progress section of the 17th IEEE Conference on Emerging Technologies and Factory Automation (ETFA 2012).

3.1 Application Model

We consider a set of mixed-criticality applications. Each application has a SIL-level, from SIL_4 (most critical) to SIL_0 (non-critical) and is developed according to the certification requirements for the particular SIL. The application model presented in this section is referring to applications scheduled with FPS and is a simplification of the application model from [TSP11a].

We consider that the mapping of tasks to processing elements (PEs) and the assignment of tasks to partitions is given, and can be determined using the approach from [TSP11a]. We model an application $\mathcal{A}$ as a directed, acyclic graph $\Gamma(\mathcal{V}, \mathcal{E})$. Each node $\tau_i \in \mathcal{V}$ represents one task. An edge $e_{ij} \in \mathcal{E}$ represents a precedence relationship between τ_i and τ_j, and indicates that τ_i must complete its execution before τ_j. Each task τ_i is characterized by a worst-case execution time (WCET) C_i (on the PE that it's assigned to for execution), a best-case execution time C_i^{min}, and in case the task uses shared resources, a maximum blocking time B_i. Additionally, τ_i has an unique priority denoted by $prio(\tau_i)$, an offset Φ_i and a maximum release time jitter J_i. Thus, considering that the graph Γ to which τ_i belongs is triggered by an external event arriving at t_0, τ_i arrives at time $t_0 + \Phi_i$ and is released after an additional maximum delay of J_i. For each application we have a deadline $D_{\mathcal{A}}$ and a period $T_{\mathcal{A}}$.

3.2 System Model

We consider architectures composed of a set $\mathcal{N}$ of PEs connected by a broadcast communication channel. We assume that the hardware and software architecture implements a temporal- and space-partitioning scheme similar to IMA [TSP11a].

Each application $\mathcal{A}_i$ is allowed to execute only within its defined partition P_j. Each partition can use its own scheduling policy. On a processing element N_i, a partition P_j is defined as the sequence P_{ij} of k partition slices p_{ij}^k, $k \geq 1$. A partition slice p_{ij}^k is a predetermined time interval in which the tasks of application $\mathcal{A}_j$ mapped to N_i are allowed to use the PE. All the slices on a processor are grouped within a Major Frame (MF), that is repeated periodically. The period T_{MF} of the major frame is given by the designer and is the same on each node. Several MFs are combined together in a system cycle that is repeated periodically, with a period T_{cycle}.

In Figure 3.1 we have 3 applications, $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$, implemented on 2 PEs,

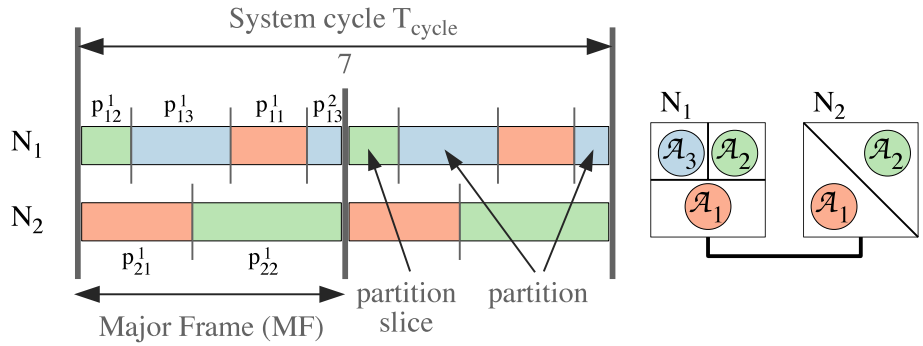


Figure 3.1: Partitioned architecture

N_1 and N_2, with $T_{MF} = 10$ and $T_{cycle} = 20$. The tasks of $\mathcal{A}_1$, for example, can execute only in partition P_1 on PE N_1, composed of the partition slice $p^1_{1,1}$, and in partition P_1 on PE N_2, composed of the slice $p^1_{2,1}$. The sequence and length of the partition slices in a MF are the same (on a given PE), but the contents of the slices can differ. An application can extend its execution over several MFs.

We don't consider communication in our analysis. However, researchers have shown how realistic bus protocols, e.g., FlexRay [PPE+08] or TTP, can be integrated into the analysis.

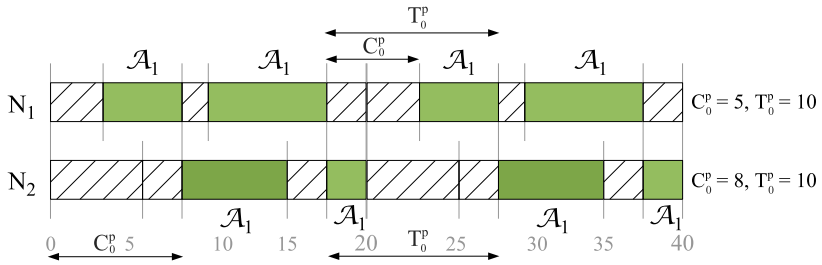
3.3 Motivational Example

Let us illustrate the importance of accurately taking the partitions into account during the analysis using the example in Figure 3.2. We consider a system with 2 PEs and 6 tasks. The partition table is given in Figure 3.2a, where two consecutive MFs with $T_{MF} = 20$ are presented. The details of the application are given in Figure 3.2b (the tasks are sorted according to their priorities–highest being on top–and the deadlines are equal to the periods).

The mapping and the partitioning are given, and we assume they are derived with our approach from [TSP11a]. The hatched partition slices represent partitions on which the application $\mathcal{A}_1$ is not allowed to execute. The partition slices corresponding to application $\mathcal{A}_1$ are colored in green.

We are interested to determine the worst-case response times of the tasks in Figure 3.2b considering the partitions in Figure 3.2a. We have compared the SA analysis from [AW96] with our proposed analysis, SA+. To facilitate the

(a)



(b)

		C_i	T_i	PE	R_i^{SA}	R_i^{SA+}
	τ_1	2	15	N_1	12	7
	τ_2	8	40	N_1	20	19
$\mathcal{A}_1$	τ_3	3	30	N_1	30	27
	τ_4	3	30	N_2	19	14
	τ_5	4	40	N_2	50	18
	τ_6	8	100	N_1	120	59

Figure 3.2: Motivational example

comparison, we consider the assumptions from [AW96], i.e., deadlines are equal to the periods, and we ignore the dependencies. The SA results are presented in column 5 in Fig. 2b, and the SA+ results are in column 6. As we can see, SA is much more pessimistic (the application is considered unschedulable) compared to our proposed analysis, SA+.

The pessimism of SA comes from its limiting assumptions that the partition slices have to be periodic within a MF. This assumption is not true in practice, but it simplifies the analysis. When analyzing the tasks in a partition P_j on a PE N_i, SA merges all the other partition slices into a "higher priority task" with WCET C_0^p (the length of the other slices) and period T_0^p. In order to apply SA in the general context of Fig. 3.2a, we have to consider C_0^p as the longest time-interval of continuous partition slices $\notin P_j$ on N_i, and T_0^p as the shortest inter-arrival time of these intervals. These values are calculated for each PE. For example, the values of C_0^p and T_0^p for N_1 and N_2 in Fig. 3.2a are as depicted in the figure. Our proposed analysis, SA+, does not assume that the partition slices have to be periodic, and thus reduces the pessimism of SA by accurately taking into account the exact position and size of the partition slices.

3.4 Response Time Analysis

WCDOPS+ [Red04] is an algorithm which performs worst-case response time analysis on fixed priority scheduled tasks disposed in tree-shaped transactions taking into consideration the precedence constraints between them, which later was extended to consider graphs [KM07].

The WCDOPS+ response time analysis for a certain task τ_{ab} is based on finding the contributions from each transaction in the system to a busy period of τ_{ab}. The busy period (also called busy window) of τ_{ab} is defined as the longest interval of time during which tasks with priority greater or equal than τ_{ab} are executed continuously [Fid98].

When studying the contribution of a transaction Γ_i to the worst case response time of a task τ_{ab}, it may be useful to somehow group the tasks of Γ_i and treat each group as if it were a single task. The creation of these groups of tasks is accomplished by defining the concepts of H sections and H segments. Two tasks of a transaction Γ_i belong to the same H section for the analysis of τ_{ab} if they belong to $hp_i(\tau_{ab})$ and there is no intermediate task in the transaction that belongs to $lp_i(ab)$. Similarly, two tasks of a transaction Γ_i belong to the same H segment for the analysis of τ_{ab} if they belong to $hp_i(\tau_{ab})$ and there is no intermediate task in the transaction that does not belong to $hp_i(ab)$. The set of tasks $hp_i(ab)$ represents the tasks belonging to a transaction Γ_i that are executed on the same PE as τ_{ab} and have a priority greater or equal than τ_{ab}. The set $lp_i(\tau_{ab})$ contains tasks belonging to Γ_i and executed on the same PE as τ_{ab} that have lower priorities than τ_{ab}.

H segments are more restrictive than H sections in the sense that if two tasks belong to the same H section with respect to a task τ_{ab}, then they *may* belong to the same τ_{ab} busy period, while if two tasks belong to the same H segment with respect to a task τ_{ab}, then they *must* belong to the same τ_{ab} busy period.

WCDOPS+ analyzes separately the contributions to the τ_{ab} busy period: first, the contributions made by all the transactions to which the task τ_{ab} doesn't belong to; secondly, the contribution made by the transaction Γ_a of which τ_{ab} is a part of. For each transaction, two types of contributions are considered: a non-blocking interference (W_i) and a blocking interference (WB_i). Since, as shown in [Red04] only one blocking H segment can contribute to the busy period, WCDOPS+ allows this contribution to the transaction with the maximum interference increase ($\Delta W = WB_i$ - W_i).

The worst-case response time of an instance of τ_{ab} with the index p_{ab} is determined by WCDOPS+ based on its completion time, $w_{abc}(p_{ab})$. The completion

time is composed of the maximum blocking time from lower priority tasks, a blocking interference and a non-blocking interference from the transactions in the system. The blocking interference is expressed through the interference increase which has to be maximized for the calculation of worst case completion time. Since there can only be one blocking segment executing in a busy period, the interference increase is chosen to be the maximum from the interference increases calculated separately:

$$\Delta W^*_{ac}(\tau_{ab}, w, p_{ab}) = MAX(\Delta W_{ac}(\tau_{ab}, w, p_{ab}), \Delta W^*_i(\tau_{ab}, w, p_{ab}, \tau_{ac})) \quad (3.1)$$

The non blocking interference is obtained by summing up the non blocking interferences from all transactions in the system, so the completion time $w_{abc}(p_{ab})$ is

$$w_{abc}(p_{ab}) = B_{ab} + W_{ac}(\tau_{ab}, w, p_{ab}) + \sum_{\forall i \neq a} W^*_i(\tau_{ab}, w, \tau_{ac}) + \Delta W^*_{ac}(\tau_{ab}, w, p_{ab}) \quad (3.2)$$

This equation is solved iteratively and because the instance p_{ab} of task τ_{ab} arrives at $\varphi_{abc} + (p_{ab} - 1)T_a$, its response time is [Red04]:

$$R^w_{abc}(p_{ab}) = w_{abc}(p_{ab}) - \varphi_{abc} - (p_{ab} - 1)T_a + \phi_{ab} \quad (3.3)$$

The worst-case response time R^w_{ab} for the task τ_{ab} is the maximum value of the result in Eq. 3.3, considering all the critical instants initiated by higher priority tasks and by τ_{ab} as well as all the job instances.

3.4.0.1 Extending WCDOPS+ with Partitioning

We have extended WCDOPS+ to take into account the partitions by using the concepts of *availability* and *demand*, inspired by the approach presented in [PPEP08]. Informally, the *availability* associated to a task τ_{ij} during a time interval t, denoted as $A_{ij}(t)$, is equal to the processor time that is not used by other partitions during t. The *demand* for a task τ_{ij} during a time interval t, denoted as $H_{ij}(t)$, is equal to the sum of the processor times required by τ_{ij} and all higher priority tasks mapped to the same processor during t.

In more general terms, the demand of a task scheduled in a partition P_k is equal to the length of its busy period when there wouldn't be any time partitions considered or when P_k would be the only partition on the processor. Fig. 3.3 shows that the demand of task τ_i during the busy window w_i is equal to the sum of the worst-case execution time of the higher priority tasks C_a and C_b and

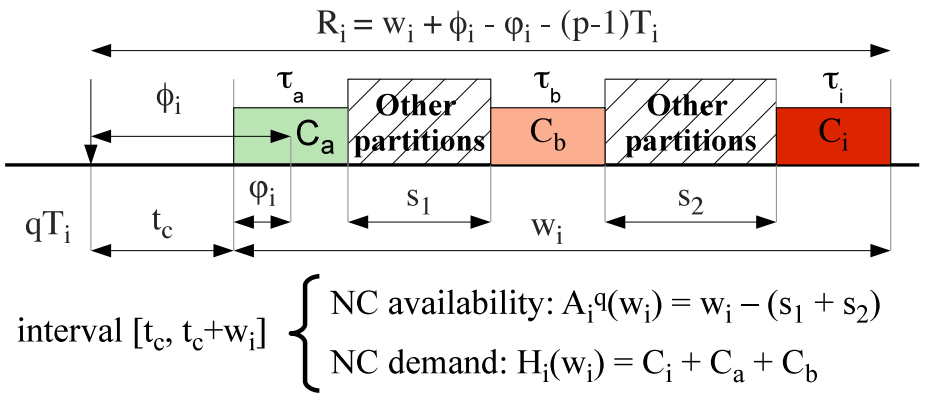


Figure 3.3: Availability and demand

the worst-case execution time of the task in question, C_i. In the WCDOPS+ analysis, the length of a τ_{ab} busy period is called τ_{ab}'s completion time and is expressed in Eq. 3.2. Thus, the demand $H_{abc}(p_{ab})$ of an instance p_{ab} of a task τ_{ab} during a busy period initiated by a task τ_{ac} is equal to p_{ab}'s completion time.

$$H_{abc}(p_{ab}) = w_{abc}(p_{ab}) \tag{3.4}$$

The availability associated to an instance p_{ab} of a task τ_{ab}, scheduled in a partition P_k, is the processing time available during $w_{ab}(p_{ab})$ for P_k. Because of the time partitioning scheme and because task τ_i can execute only during its own partition P_k, the availability is calculated by subtracting from $w_{ab}(p_{ab})$ the time reserved for the "other" partitions. In Fig. 3.3, the availability is shown as what is left after subtracting from w_i the durations of the other partitions, s_1 and s_2.

As a consequence of considering the partitioning scheme, the completion time $w_{abc}(p_{ab})$ of an instance p_{ab} of task τ_{ab} is replaced by an *extended completion time*, $e_{abc}(p_{ab})$, computed according to Algorithm 1. The purpose of this algorithm is to increase a task's completion time until the availability is at least as large as the demand during this time interval. The algorithm starts by initializing the extended completion time of p_{ab} with p_{ab}'s original completion time and the availability and demand with 0 (lines 10–12). It then proceeds to iteratively re-compute the availability and demand until the availability is greater or equal to the demand (lines 13–19). At each iteration, the extended completion time of p_{ab} is increased with the difference between the current values of the availability and demand (lines 16–18). Finally, the obtained extended completion time is returned (line 20).

Algorithm 1 EXTENDED_COMPLETION_TIME

Inputs:
τ_{ab} - a task;
p_{ab} - an instance of τ_{ab};
τ_{ac} - a task starting a p_{ab} busy period;
w_{abc} - completion time of p_{ab};
Outputs:
e_{abc} - extended completion time of p_{ab};

begin
$e_{abc} \leftarrow w_{abc}$;
$demand \leftarrow 0$;
$availability \leftarrow 0$;
repeat
  $demand \leftarrow COMPUTE\_DEMAND(\tau_{ab}, p_{ab}, \tau_{ac}, e_{abc})$;
  $availability \leftarrow COMPUTE\_AVAILABILITY(w_{abc})$;
  if $demand > availability$ **then**
    $e_{abc} \leftarrow e_{abc} + demand - availability$;
  end if
until $availability \geq demand$
return e_{abc};
end

Acknowledgments The solution to the problem of extending WCDOPS+ to consider partitioning was found by me together with my colleague Vlad Acretoaie. I would like to acknowledge here his contribution.

3.5 Evaluation

Table 3.1 presents our experimental evaluation. We have used seven synthetic benchmarks and one real-life example. The synthetic benchmarks were generated similar to [TSP11a], and the number of PEs and number of tasks in the FPS applications are presented in columns 2 and 3, respectively, in Table 3.1. The real-life case study is derived from the "automotive" benchmark in the E3S suite [Dic]. The partition tables and the mapping have been generated using the "Initial Solution" approach from [TSP11a]. We have run both SA and SA+ on these tasks sets. Columns 4 and 5 present the number of tasks found schedulable, i.e., $R_i \leq D_i$, using SA and SA+, respectively. Columns 6 and 7 present the minimum and, respectively, maximum percentage reduction of worst-case response times obtained with SA+ compared to SA for the tasks in the bench-

mark. To show the overall reduction in the pessimism of SA+ over SA, the last column in the table represents the percentage reduction of worst-case response times obtained with SA+ compared to SA averaged over all tasks.

As we can see from these results, accurately taking into account the partitions during the analysis can significantly reduce the pessimism of the results.

Table 3.1: Experimental results

Test Case	PEs	Tasks	SA	SA+	Min % reduction	Max % reduction	% reduction
1	2	4	3	4	41.66	68.29	59.04
2	3	7	5	7	0	64	27.95
3	3	10	6	10	21.05	71.35	46.82
4	4	16	12	15	30.11	78.06	41.66
5	4	19	17	19	8.43	59.34	24.91
6	5	22	20	22	25.42	70.15	56.67
7	5	25	21	25	12.58	59.21	30.23
automotive	3	5	4	4	13.81	44.44	34.18

Chapter 4

TTEthernet Analysis and Simulation

This chapter presents methods for performing worst-case delay analysis for the RC messages in TTEthernet and AFDX networks. We will start by presenting a state-of-the-art analysis for TTEthernet and continue with analyses for AFDX. Out of the presented methods for AFDX there is none with a direct and complete applicability to TTEthernet, but these analyses are relevant since they provide a starting point for new analyses for TTEthernet. We will focus in this regard on the so-called "trajectory approach" which we present extensively and for which we propose an extension to make it more suitable for being applied to TTEthernet networks.

In its last part, this chapter also presents a simulator for the behavior of a TTEthernet network. The simulator is implemented to mimic the network's functioning as it was described in section 2.2.5 and has the purpose to obtain worst-case end-to-end delays for the RC traffic in the network. We will compare the results of the simulations with the results analytically derived with the method described in section 4.1.

4.1 TTEthernet Analysis

The integration of TT and RC traffic is a fundamental problem in a TTEthernet network used in mixed-criticality systems. In [Ste10a], Steiner proposes a solution to synthesize static TT schedules based on a SMT-solver, but ignores the RC traffic. In [Ste10c, Ste10b], Steiner proposes an analysis for rate-constrained traffic. Later, in [Ste11], these approaches are integrated. We are interested here in the analysis of the rate-constraint traffic, but for a complete understanding we need to refer to some concepts related to the synthesis of static schedules for TT communication.

When generating the static schedules one thing that should be avoided is to have time-triggered messages scheduled back to back, since this can lead to a starvation problem for the rate-constrained traffic [Ste11]. That is why, in [Ste11], Steiner introduces the concept of "schedule porosity" as a characteristic of the TT schedule which measures whether the rate-constrained traffic has enough bandwidth for reasonable functioning in terms of latency and jitter. In simpler terms, achieving schedule porosity is about having enough free (blank) intervals in the TT schedule in which the RC frames can be transmitted [Ste11]. In the rate-constrained traffic analysis from [Ste11] that we present, the assumption is that the TT schedule porosity is obtained by alternating the TT and RC transmission slots (a TT slot has the length l_{TT}, while a blank slot reserved for RC transmission has the length l_{blank}).

We consider a network composed of one switch (denoted k) and n end systems (denoted $v_i, i = 1 \dots n$). The transmission speed is considered to be equal on all the physical links. We already established that the physical link connecting an end system with the switch corresponds to two dataflow links. Thus, from the perspective of an end system v_i, $[v_i, k]$ is an egress dataflow link and $[k, v_i]$ is an ingress dataflow link. From the perspective of switch k, the egress link is $[k, v_i]$ and the ingress link is $[v_i, k]$.

The maximum burst on a dataflow link (denoted by $burst^{[v_x,k]}$) equals the sum of all the frame instances that are transmitted on that link. We can split the burst on the ingress dataflow links in respect to the egress dataflow links. This way, we can express the fraction of the data that is received by the switch on an ingress link which will be transported on an egress link. We denote by $burst^{[v_x,k]}_{[k,v_y]}$ the fraction of the ingress burst on the link $[v_x, k]$ that will be forwarded by the switch on the egress link $[k, v_y]$ (Equation 4.1).

$$burst^{[v_x,k]}_{[k,v_y]} = \sum_i \begin{cases} f_i^{[v_x,k]}.length & if \exists f_i^{[k,v_y]} \\ 0 & otherwise \end{cases} \tag{4.1}$$

Considering the egress link $[k, v_y]$, the sum of all the $burst^{[v_x,k]}_{[k,v_y]}$ is denoted by $BURST^{in}_{k,v_y}$. The fraction of $burst^{[v_x,k]}_{[k,v_y]}$ concerning only the RC frames is denoted by $\widehat{burst}^{[v_x,k]}_{[k,v_y]}$ and expressed in Equation 4.2. Similarly to $BURST^{in}_{k,v_y}$, but involving only RC frames, we define $\widehat{BURST}^{in}_{k,v_y}$ as the sum of $\widehat{burst}^{[v_x,k]}_{[k,v_y]}$ for all the ingress dataflow links to the switch.

$$\widehat{burst}^{[v_x,k]}_{[k,v_y]} = \sum_i \begin{cases} rc_i^{[v_x,k]}.length & if \exists rc_i^{[k,v_y]} \\ 0 & otherwise \end{cases} \tag{4.2}$$

The RC frames arriving at the TTEthernet switches are stored in queues (a queue for each egress dataflow link). The queues are serviced in FIFO order. This means that the latency of a RC frame f_i which entered the switch on the dataflow link $[v_x, k]$ and will leave the switch on the dataflow link $[k, v_y]$ depends on the size of the queue corresponding to the egress link $[k, v_y]$ and on the link's transmission speed (denoted by $wirespeed$). For obtaining the maximum latency we need to maximize the backlog.

The backlog is given by the difference between the maximum ingress dataflow and the egress dataflow [Ste11]. The maximum backlog on the egress dataflow link $[k, v_y]$ is given then by Equation 4.3. From this point it is straightforward to derive the maximum latency for a frame in the switch as the time necessary for the transmission of the maximum backlog and of the frame itself (see Equation 4.4).

$$Q^{out}_{[k,v_y]} = BURST^{in}_{k,v_y} - max(burst^{[v_x,k]}_{[k,v_y]}) \tag{4.3}$$

$$max(latency^{[v_x,k]}_{[k,v_y]}) = \frac{Q^{out}_{[k,v_y]} + f_i.length}{wirespeed} \tag{4.4}$$

Taking into account the TT schedule porosity manifested as the alternation of time-triggered and rate-constraint transmission slots (with lengths l_{TT} and l_{blank}, as previously defined), we need to adjust the calculation of the maximum latency. The maximum outgoing queue size for a RC frame is given by

Equation 4.5 and the maximum latency is given by Equation 4.6.

$$\widehat{Q}^{out}_{[k,v_y]} = \widehat{BURST}^{in}_{k,v_y} - max(\widehat{burst}^{[v_x,k]}_{[k,v_y]}) + l_{TT} \times \left(\left\lfloor \frac{max(\widehat{burst}^{[v_x,k]}_{[k,v_y]})}{l_{TT} + l_{blank}} \right\rfloor + 1 \right) \tag{4.5}$$

$$max(\widehat{latency}^{[v_x,k]}_{[k,v_y]}) = \frac{\widehat{Q}^{out}_{[k,v_y]} + f_i.length + \left\lceil \frac{\widehat{Q}^{out}_{[k,v_y]}}{l_{blank}} \right\rceil \times l_{TT}}{wirespeed} \tag{4.6}$$

4.2 AFDX Analysis

To provide deterministic upper bounds for the end-to-end delays of the transmitted avionics flows is a requirement for the certification of AFDX networks. In the beginning of this section we will make a short survey of the methods used to obtain these deterministic upper bounds. We will continue by presenting in more detail the most widely employed methods for the problem in question: network calculus and trajectory approach.

Exact worst-case end-to-end delays can be obtained with the model-checking approach described in [CSEF06]. The AFDX network is modeled there using timed automata and its properties are verified using UPPAAL. Since in this approach all the possible states of the system are explored, the resulted worst-case end-to-end delays are exact. This solution is suitable only for small AFDX configurations. For realistic network configurations the models used in [CSEF06] will generate a combinatorial explosion and they will be impossible to solve. Besides the mentioned model-checking approach, [CSEF06] also describes a simulation approach for obtaining the AFDX flows' worst-case latencies. Different parameters related to the frames' emission are varied in order to evaluate their influence on the end-to-end delays. However, this approach, although useful for estimating the network load, does not provide guarantees for the frames' latencies.

Another approach that gives exact worst-case latency is presented in [ASF11]. The size of the AFDX configurations for which this approach is suitable has increased a couple of times compared to [CSEF06], to up to 50 virtual links, but it is still not enough for realistic AFDX networks (which can have more than 1000 virtual links).

A holistic analysis for AFDX-based distributed systems is described in [GPGH11, GPGH12]. The proposed analysis deals with the whole system, that's why the term holistic is used to characterize it. In a holistic approach, an end-to-end flow is generated by a periodic sequence of external events and includes the tasks generating the messages that are sent through the AFDX network [GPGH12]. The same authors demonstrate in [RGPH11] that different response time analyses for different resources (like the processor or the communication networks) of a distributed system can be composed if these analyses can handle offsets and input jitter for the triggering events [GPGH12]. The proposed response time analysis for AFDX messages from [GPGH12] accounts for offsets and jitters in the message flows. Using the composition mechanisms from [RGPH11], this analysis could be integrated with response time analyses for processing units in a holistic analysis of a distributed system.

4.2.1 Network Calculus

Network Calculus [Cru91, LBT01] is a theoretical framework which can be used to solve flow problems in networks. Network calculus can take into account different types of constraints typically imposed on traffic flows in a network, such as the network links' transmission rates, congestion control and traffic shapers. In the context of AFDX, the Network Calculus has been useful for determining upper bounds for the end-to-end delays for certification purposes (namely, for the certification of the avionics network of Airbus A380) [GFF03, FFG06].

One of the fundamental concepts in Network Calculus is the cumulated traffic (R), that is the number of bits emitted by the considered flux from the beginning. So, the data flows can be described by the means of the cumulative function $R(t)$, which represents the number of bits emitted in the interval $[0, t]$ [LBT01].

The constraints on the data flows are modeled by arrival curves, noted with α. The arrival curves limit the amount of bits that could be emitted by the flow in a certain interval δ. The network elements are modeled through the concept of service curve, noted with β. Once the arrival curve α for a flow and the service curve β for a network element are known, one can calculate an upper-bound on the delay experienced by each bit passing through the considered network element.

One of the major advantages of Network Calculus is that it allows the propagation of results from one element of the network to the next one. More precisely, if we know the arrival curve α for a flow which traverses a network element characterized by the service curve β, we can determine the arrival curve α' which constrains the output flow, where $\alpha' = \alpha \oslash \beta$ and the meaning of the $\oslash$ operator

is given by the following equation:

$$\alpha \oslash \beta(t) = \sup_{u \geq 0}(\alpha(t+u) - \beta(u)) \tag{4.7}$$

In the context of AFDX, the arrival curves characterizing the flows in the system are defined based on the VL parameters. The traffic emitted on a VL is limited by an arrival curve expressed as:

$$\gamma_{s_{max}/BAG, s_{max}}(t) = s_{max} + (s_{max}/BAG) \times t \tag{4.8}$$

Regarding the service curves for the AFDX network elements, the switch output port is characterized by a service curve β. The network links are considered elements that don't modify the traffic (the electromagnetic transmission delay is insignificant). The end systems are modeled in reception by a simple buffer and, in emission, their output is considered a flow which enters the network.

Besides obtaining upper bounds on the end-to-end delays of flows, by applying Network Calculus, one can also obtain information on the latency time in the switch output ports [BSF09]. This is useful, because based on this information, the switch memory buffers can be dimensioned so that buffer overflows are avoided.

The Network Calculus is a holistic approach. A holistic approach is one that considers the worst case scenario on each node visited by a flow and takes into account the maximum jitter introduced by the previous visited nodes. This type of approach is considered pessimistic since it can lead to impossible scenarios. Therefore, Network Calculus is a pessimistic method for the previously mentioned reason and also because in Network Calculus envelops are used instead of the precise arrival and service curves [BSF09].

Network Calculus was extensively used for the timing analysis of AFDX networks and, in this context, several improvements appeared along the way. We will briefly mention some of them. One such improvement is the "grouping technique" introduced in [GFF03] and later also applied in the trajectory approach in [BSF09, BSF12]. The flows that go through the same switches (that is, they have at least two common dataflow links) are grouped together. The packets at the exit of the first switch are serialized and they will not delay each other in the next switches. By taking into account this serialization of packets, this technique obtains tighter upper bounds for the end-to-end delays.

Heuristic optimization methods (such as genetic algorithms and alpha-beta search), in conjunction with Network Calculus, were applied in [FFG06]. A probabilistic analysis for the end-to-end delays' upper bounds using stochastic Network Calculus is presented in [SRF09]. The extension of the Network Calculus method to consider the scheduling of periodic flows was also considered

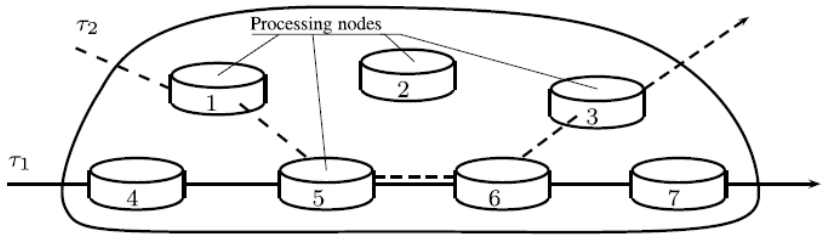


Figure 4.1: The network model in the trajectory approach [Mar04]

by the researchers [LSF10]. The scheduling was modeled by assigning offsets to the periodic flows. The analysis in [LSF10] works with any offset assignment algorithm.

4.2.2 Trajectory Approach

The trajectory approach [Mar04, MM05, MM06b] was designed to provide quantitative and deterministic Quality of Service (QoS) guarantees for real-time flows in a packet switching network [MM05]. More precisely, the two QoS parameters for which the trajectory approach is able to provide upper bounds are the jitter and the worst-case end-to-end response times of flows. The flows considered by the trajectory approach are sporadic and there is no assumption regarding the arrival time of packets.

The trajectory approach was successfully used to determine the worst-case end-to-end response delays of flows in AFDX networks [Bau11, BSF09, BSF12] giving better results than the network calculus. This method is relevant also for the TTEthernet protocol and we will study its applicability to TTEthernet networks. We will focus on the "original" trajectory approach as it was presented in [Mar04, MM05, MM06b] and not on the versions of trajectory approach optimized for AFDX networks introduced in [Bau11, BSF09, BSF12].

4.2.2.1 Models and notations

The general architecture of a network as it is considered by the trajectory approach is illustrated in Figure 4.1. We can see that the network is composed of interconnected processing nodes crossed by flows.

The trajectory approach considers the worst case scenario experienced by a packet on its trajectory. The worst case scenario is considered for the whole trajectory and not on any node visited. But before going into more details, let's briefly present the models considered by the trajectory approach, as in [MM06a]:

- The scheduling model: in every node of the network the packets are scheduled based on their fixed and dynamic priorities (a FP/DP* policy as it is named in [Mar04]). The first criterion used when scheduling packets is their fixed priority (FP). The dynamic priority (DP) is the secondary criterion, employed only for the packets having the same fixed priority. In the trajectory approach, the dynamic priority of a packet is calculated on the first visited node (this is marked by the star in DP*). The packet scheduling is assumed to be non-preemptive. This means that a node can send a packet only after the current packet transmission (if any) has ended.
- The network model: it is assumed that the network links are FIFO and that the network delay between two nodes is bounded by L_{min} and L_{max}. It is assumed that there are neither packet losses nor network failures.
- The traffic model: the trajectory approach considers a set $\{\tau_1, ..., \tau_n\}$ of n sporadic flows. Each flow τ_i is following a fixed path (denoted by $\mathcal{P}_i$). A path is an ordered sequence of nodes, the first node in the sequence being the ingress node of the flow. In Figure 4.1 we have two flows τ_1 and τ_2, with τ_1 following path $\mathcal{P}_1 = \{4, 5, 6, 7\}$ and τ_2 following path $\mathcal{P}_2 = \{1, 5, 6, 3\}$.

It is assumed in the trajectory approach that any two flows τ_i and τ_j following distinct and intersecting paths ($\mathcal{P}_i \neq \mathcal{P}_j$ and $\mathcal{P}_i \cap \mathcal{P}_j \neq \emptyset$) can not cross each other twice. That is, there can be only one common portion between their paths. In the cases where this assumption is not verified, the solution is to decompose the flows violating the assumption into as many independent flows as necessary to meet the assumption. Practically, if flow τ_j violates the assumption in regard to path $\mathcal{P}_i$ of τ_i, then flow τ_j will be considered as a new flow after leaving $\mathcal{P}_i$. This decomposition is done repeatedly until all the flows in the network respect the assumption.

In the case of the flows following intersecting paths, it is useful to denote the first and the last nodes of the common path section. Thus, in [Mar04, MM05] the following notations are introduced: $first_{i,j}$ and $last_{i,j}$ are the first and, respectively, the last nodes visited by τ_j on the path $\mathcal{P}_i$ of τ_i. Analogue meanings are reserved for $first_{j,i}$ and $last_{j,i}$. An illustration of these notations is provided in Figure 4.2 (taken from [MM05]) for both possible flow orientations.

A sporadic flow τ_i is defined by the following properties:

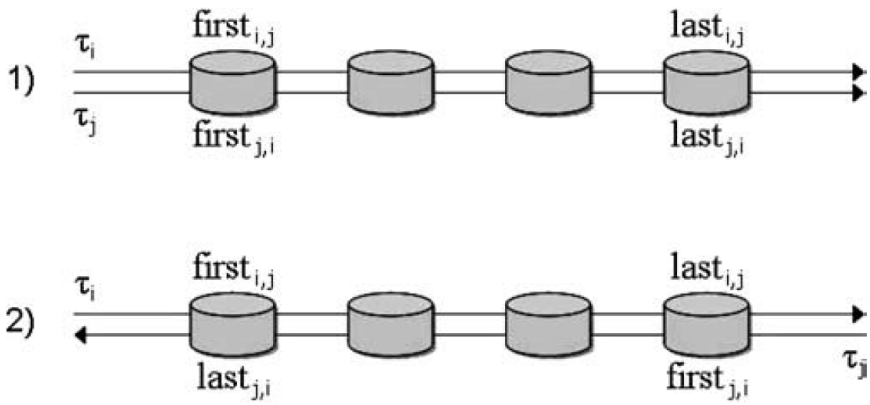


Figure 4.2: $first_{i,j}$, $last_{i,j}$, $first_{j,i}$ and $last_{j,i}$ [MM05]

- T_i is the minimum inter-arrival time between two successive packets at the ingress node of the flow
- P_i is the static priority of τ_i
- C_i^h is the maximum processing time on the node h (h $\in \mathcal{P}_i$) of a packet of τ_i
- J_i is the maximum release jitter (at the ingress node) of packets of τ_i
- D_i is the maximum end-to-end response time acceptable for a packet of τ_i

As mentioned above, in the FP/DP* scheduling scheme proposed by the trajectory approach the fixed priority of packets (inherited from the flows the packets belong to) is the principal criterion. It makes sense then to divide the flows according to their fixed priority levels. In [Mar04], in regard to each flow τ_i, the following sets are introduced:

- $hp_i = \{j \in [1,n], P_j > P_i\}$ - the set of flows with a (strictly) higher fixed priority than the priority of τ_i
- $sp_i = \{j \in [1,n], j \neq i, P_j = P_i\}$ - the set of flows having the same fixed priority as τ_i
- $lp_i = \{j \in [1,n], P_j < P_i\}$ - the set of flows having a (strictly) lower fixed priority than this of τ_i

From the multiple flavors of FP/DP* that can be employed (like FP/FIFO, FP/EDF or FP/WFQ), taking into account the specifics of the AFDX and

TTEthernet networks, in this thesis we will consider only FIFO (when all flows have the same static priority) and FP/FIFO.

4.2.2.2 Study of a packet's trajectory

The problem solved by the trajectory approach is to determine the worst case end-to-end response times of a set of sporadic flows. The solution needs, of course, to be general enough so it doesn't depend on the arrival times of the packets in the network. The time is considered to be discrete. This simplifying assumption doesn't affect the generality of the solution since it was proven in [BHR90] that the results obtained with discrete scheduling are as valid as those obtained with continuous scheduling if the flow parameters are multiples of the node clock tick.

The end-to-end response time of a packet has two components: the time spent in each node the packet has crossed and the time spent by the packet "traveling" on the network links. We already said that the transmission delay on a network link is upper-bounded by L_{max}. Then, since we are interested in the worst case response times, the delay on the network links can be easily maximized using L_{max}. The second component (the time spent in the nodes) is more complicated to analyze.

The non-preemptive scheduling of packets considered in the trajectory approach has the effect that once the transmission of m is started it can not be interrupted. Since the processing time of m in each node of the path is easy to obtain, we need to determine only the latest starting time of m on it last visited node.

This is done in the trajectory approach by moving backwards through m's trajectory (from the last visited node to the ingress node), in each visited node identifying the busy period and the preceding packets impacting the end-to-end response time of m.

The busy period concept may need a bit more detailed explaining: "a busy period of level L is an interval $[t, t')$ such that t and t' are both idle times of level L and there is no idle time of level L in (t, t'). An idle time t of level L is a time such as all packets with priority greater than or equal to L generated before t have been processed at time t" [MM06a, MM05]. So the busy period is an interval of time in which the processing node is continuously busy.

In the following, we will exemplify the principles of trajectory approach first from a theoretical perspective (as in [MM06a, MM05, MM06b]) and, later on, from a more concrete perspective (by applying the trajectory approach to a

simple AFDX network).

Let's consider the node q, found on the trajectory of the packet m. Let bp^q be the busy period in which m is processed on node q. We refer to the first packet processed in the busy period bp^q as $f(q)$. Since $f(q)$ belongs to bp^q, $f(q)$ has a priority higher or equal to this of packet m. It is not mandatory that packet $f(q)$ comes from the previous node in the trajectory, $q-1$. $f(q)$ can belong to a flow following a different path than P_i.

We reiterate that the trajectory approach strategy is to move backwards through the sequence of nodes m has visited and to identify the busy periods and the preceding packets that have an impact on the delay of m. This is why a new packet needs to be considered: $p(q-1)$. With $p(q-1)$ we denote the first packet processed between $f(q)$ and m on node q which comes from node $q-1$. On node $q-1$, the packet $p(q-1)$ was processed in a busy period of level corresponding to the priority of $p(q-1)$; busy period which we denote bp^{q-1}.

We can perform on node $q-1$ an analysis analogue to the one applied on node q. Let $f(q-1)$ be the first packet processed in the busy period bp^{q-1} with a priority higher or equal to this of packet $p(q-1)$. $f(q-1)$ may have not come from node $q-2$, so we define $p(q-2)$ as the first packet processed between $f(q-1)$ and m on node $q-1$ and coming from node $q-2$.

This reasoning needs to be applied repeatedly until we reach the ingress node of the flow, that is, until we consider the busy period of level corresponding to the priority of $p(1)$ in which the packet $f(1)$ is processed. When we are done, the busy periods that will be used in the computation of the end-to-end response time of m have been determined. In Figure 4.3 we illustrate the decomposition strategy employed by the trajectory approach that we just described.

Let a_m^h be the arrival time of packet m on node h. The arrival time of packet $f(1)$ at node 1 can be considered the time origin ($a_{f(1)}^1 = 0$). For simplicity, the packets processed on a node h between $f(h)$ and $p(h)$ are consecutively numbered [MM06a]. So, on node q, packet $(m-1)$ is the packet preceding packet m.

The latest starting time of packet m on the last node in its trajectory can be calculated using parts of the busy periods we previously analyzed [MM05]. So, the latest starting time of packet m in node q can be expressed as:

the processing time on node 1 of packets $f(1)$ to $p(1)$ + L_{max}
+ the processing time on node 2 of packets $f(2)$ to $p(2)$ + L_{max} - ($a_{p(1)}^2$ - $a_{f(2)}^2$)
+ ...

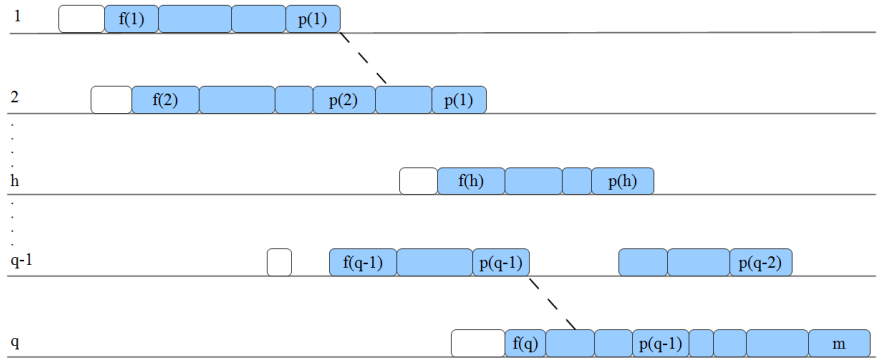


Figure 4.3: Response time of packet m

$$+ \text{ the processing time on node q of packets } f(q) \text{ to } (m-1) \text{ - } (a^{q}_{p(q-1)} \text{ - } a^{q}_{f(q)}) + \delta_i^{1,q}(t)$$

$\delta_i^{1,q}(t)$ (or, more simply, $\delta_i(t)$ or δ_i) represents the delay incurred by m directly directly due to non-preemption along its trajectory from node 1 to q [MM05]. We will come back to this term in section 4.2.2.4.

We introduce now the notation $\tau(g)$ which is used to denote the index of the flow to which the packet g belongs to. If we denote the latest starting time on node h of the packet m of flow τ_i (packet generated at time t) by $W_i^q(t)$, then, by summing up the terms above, we get:

$$W_i^q(t) = \sum_{h=1}^{q}\Big(\sum_{g=f(h)}^{p(h)} C^h_{\tau_{(g)}}\Big) - C_i^q + (q-1)\cdot L_{max} + \delta_i^{1,q}(t) - \sum_{h=2}^{q}(a^h_{p(h-1)} - a^h_{f(h)}) \tag{4.9}$$

This latest starting time can be further maximized. In [MM06a] it is proposed to annul the term $(a^h_{p(h-1)} - a^h_{f(h)})$ for all the nodes in the trajectory. The reasoning for this is based on the following observation. For any node h belonging to the path P_i of the flow τ_i, if there is no flow τ_j such that $h = first_{j,i}$, then $p(h-1) = f(h)$ [MM05]. If so, then $(a^h_{p(h-1)} - a^h_{f(h)}) = 0$. Otherwise put, $p(h-1)$ not being the same as $f(h)$ implies that there must be a flow τ_j such that h is the first node visited by τ_j on path P_i. This means that all the packets in the interval $[f(h), p(h-1))$ cross path P_i for the first time at node h, therefore their arrival times can be postponed in the busy period where $p(h-1)$ is processed [MM05]. Therefore, in the worst case scenario $p(h) = f(h+1)$ for all nodes in P_i. In these conditions, the latest starting time of packet m on node

h can be expressed as:

$$W_i^q(t) = \sum_{h=1}^{q}(\sum_{g=f(h)}^{f(h+1)} C^h_{\tau_{(g)}}) + (q-1) \cdot L_{max} - C_i^q + \delta_i(t) \tag{4.10}$$

Still, the goal is to obtain an upper bound on the end-to-end response time of any flow in the network, not on the latest starting time of packet m on node h. This is achieved in [MM06a], where it is proven that the worst case end-to-end response time of any flow τ_i can not be greater than:

$$R_i = \max_{t \geq -J_i} (W_{i,t}^{last_i} + C_i^{last_i} - t) \tag{4.11}$$

$W_{i,t}^{last_i}$ represents the latest starting time of packet m generated at time t on its last node visited and $last_i$ denotes this last visited node.

4.2.2.3 Trajectory approach applied to AFDX

After presenting the principle behind the trajectory approach, in this section we will show how the trajectory approach can be applied in the context of AFDX. The correspondence between the trajectory approach and the AFDX concepts is detailed first, as in [BSF09]. The concept of node from the trajectory approach corresponds to an AFDX switch output port (including the output link). It is important to notice that an AFDX switch does not correspond to a trajectory approach node, but each of its output ports does. The concept of flow in the trajectory approach is paired by the concept of virtual link path in AFDX. The links in the trajectory approach system correspond to the switching fabric in AFDX.

Furthermore, the following assumptions of the trajectory approach are verified by AFDX [BSF09]:

- the delay experienced by a packet on the switching fabric can be considered constant: $L_{min} = L_{max} = 16\ \mu$s
- on the AFDX networks there are neither collisions nor packet losses
- the routing of the AFDX VLs is statically defined

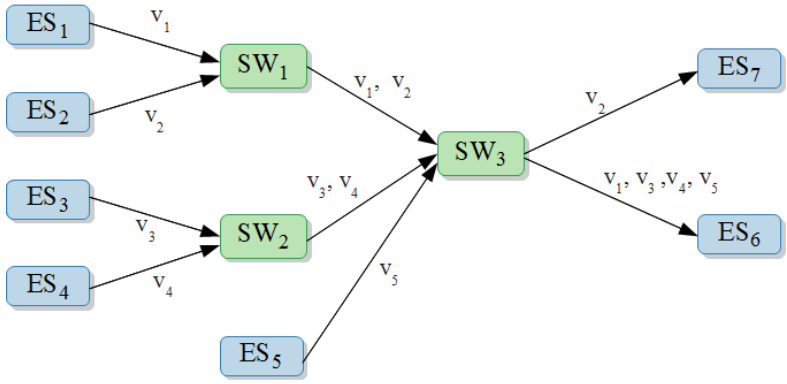


Figure 4.4: Example AFDX network

The correspondence between the VL parameters and the parameters of the sporadic flows is given next:

$$J_i = 0,\ T_i = BAG,\ C_i^h = s_{max}/R$$

The transmission rate R is constant (typically, $R = 100\ Mb/s$) for all AFDX ports, so the processing time is constant for every node h in the network ($C_i^h = C_i = s_{max}/R$).

In the following we are going to apply the trajectory approach to an example AFDX network. The system is shown in Figure 4.4 and the characteristics of the five virtual links appearing in the network are given in Table 4.1. We consider that the transmission rate is uniform across the network ($R = 100\ Mb/s$) and that the latency on the switching fabric is constant ($L = 16\ \mu s$). We emphasize that, as it can be seen in Table 4.1, for our example, $C_i^h = s_{max}/R = C = 40\ \mu s$. We consider that the packets are scheduled in this example according to the FIFO policy and that the static priorities of the corresponding trajectory approach flows are equal. The distributed system on which the trajectory approach is applied is shown in Figure 4.5. This system is the trajectory approach equivalent of our example AFDX configuration.

We are going to consider the arbitrary scheduling of the packets depicted in Figure 4.6. By convention, the packets are numbered according to their corresponding virtual link. We are interested in the starting time of packet 4 on node $SW3-ES6$. We consider the origin of time to be the arrival time of packet 4 on node $ES4-SW2$. Before packet 4 is processed on node $ES4-SW2$, we can see

VL	$BAG(\mu s)$	$s_{max}(bits)$	$C(\mu s)$
v_1	4000	4000	40
v_2	4000	4000	40
v_3	4000	4000	40
v_4	4000	4000	40
v_5	4000	4000	40

Table 4.1: Virtual links' characteristics

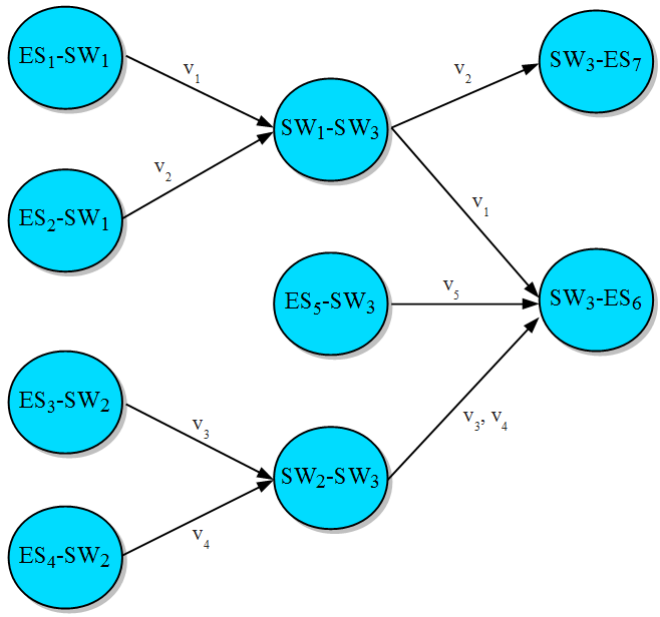


Figure 4.5: The trajectory approach system corresponding to the AFDX example

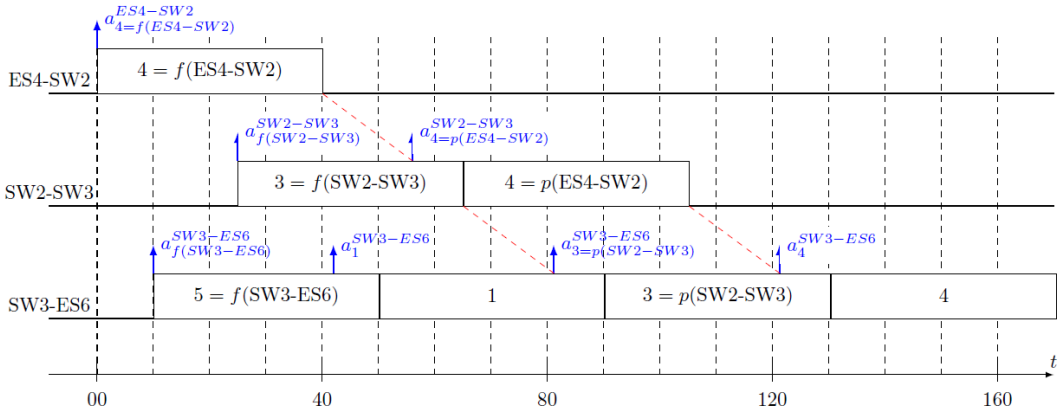


Figure 4.6: Example scheduling of packets

other packets arriving on nodes $SW2-SW3$ and $SW3-ES6$: packet 3 on node $SW2-SW3$ ($a_3^{SW2-SW3} = 25\ \mu s$) and packet 5 on $SW3-ES6$ ($a_5^{SW3-ES6} = 10\ \mu s$). After it has been processed on node $ES4-SW2$, packet 4 arrives on $SW2-SW3$ with the switching fabric delay at $a_4^{SW2-SW3} = 56\ \mu s$, when the previously arrived packet 3 is being processed. So packet 4 needs to wait until packet 3 is freeing the switch output. Taking into account the switching fabric latency, packet 4 arrives at node $SW3-ES6$ at $a_4^{SW3-ES6} = 121\ \mu s$. In the meanwhile, on node $SW3-ES6$, have arrived packet 1 (at $a_1^{SW3-ES6} = 42\ \mu s$) and packet 3 coming from $SW2-SW3$ at ($a_3^{SW3-ES6} = 81\ \mu s$).

To apply the trajectory approach we need to consider the busy periods traversed by packet 4 along its path. These busy periods need to be considered backwards (from the last visited node to the ingress node). Thus, we must start with busy period $bp^{SW3-ES6}$ - the busy period in which packet 4 is processed on node $SW3-ES6$.

We identify $f(SW3-ES6)$ to be packet 5 (belonging to v_5). Indeed, packet 5 is the first packet processed in the busy period $bp^{SW3-ES6}$ with a priority higher than or equal to this of packet 4, thus matching the definition of $f(SW3-ES6)$. Since packet 5 is not coming from the same node as packet 4, the next step would be to identify $p(SW2-SW3)$. We remind the reader that $p(SW2-SW3)$ is the first packet processed between $f(SW3-ES6)$ and packet 4, that comes from the same node as packet 4. It is clear then that $p(SW2-SW3)$ is packet 3.

Packet $p(SW2-SW3)$ was processed on node $SW2-SW3$ in the busy period

$bp^{SW2-SW3}$. For the busy period $bp^{SW2-SW3}$ we need to identify the packet $f(SW2-SW3)$ with an analogue meaning to the one of $f(SW3-ES6)$. We can see that, in the example we proposed, $f(SW2-SW3) = p(SW2-SW3)$. On node $SW2-SW3$ we identify $p(ES4-SW2)$ as being packet 4.

We move backwards to the busy period $bp^{ES4-SW2}$ of level corresponding to $p(ES4-SW2)$. For this busy period we can see that $f(ES4-SW2) = p(ES4-SW2)$.

Since we already detailed the arrival times of the packets on the different nodes, we have all the elements to determine the latest starting time of packet 4 on node $SW3-ES6$. We remind the principle of calculation: "In a node h, we count the processing times of the packets between $f(h)$ and $p(h)$ minus the difference between the arrival time of $p(h-1)$ and $f(h)$ in node h ($a^h_{p(h-1)}$ - $a^h_{f(h)}$)" [BSF09].

We apply the calculation principle for each node (we remind that the processing times are constant in the system (and denoted by C) and that the switching fabric delay is noted with L):

- for $ES4-SW2$ we have only $C_4^{ES4-SW2} = C$
- for $SW2-SW3$ we have $C_3^{SW2-SW3} - (a_4^{SW2-SW3} - a_3^{SW2-SW3}) = C - (56-25)$
- for $SW3-ES6$ we have $C_5^{SW3-ES6}+C_1^{SW3-ES6}+C_3^{SW3-ES6}-(a_3^{SW3-ES6}-a_5^{SW3-ES6}) = 3\cdot C-(81-10)$

By taking into consideration the switching fabric delay and summing up, we get the following result for the latest starting time of packet 4 on node $SW3-ES6$: $5\cdot C+2\cdot L-(a_4^{SW2-SW3}-a_3^{SW2-SW3})-(a_3^{SW3-ES6}-a_5^{SW3-ES6}) = 5\cdot C+2\cdot L-(56-25)-(81-10) = 5\cdot 40+2\cdot 16-31-71 = 130$ (μs). From this point, for finding out the end-to-end delay of packet 4 we just need to add to the previous result the processing time of packet 4 on node $SW3-ES6$. Summing up, the end-to-end delay is $130+40 = 170$ (μs).

4.2.2.4 Computation of the worst-case response time

In the beginning of the section 4.2.2 we presented the principle behind the trajectory approach and in section 4.2.2.3 we saw this principle applied to a simple AFDX network. In this section we will present in more detail how the

worst-case response times of sporadic flows are calculated using the trajectory approach. Without restricting too much from the generality of the solution and having in mind the scheduling policies that are most interesting for our thesis, we will consider in this section that the packets' scheduling is done according to a FP/FIFO algorithm.

Coming back to the latest starting time of a packet m of flow τ_i on the last node of its trajectory expressed in Equation 4.10, three components of $W_i^q(t)$ can be identified as in [Mar04, MM06b]:

- $X_i(t) = \sum_{h=1}^{q}(\sum_{g=f(h)}^{f(h+1)} C^h_{\tau_{(g)}}) - C_i^q$ - the delay caused by the packets having a priority higher or equal to m
- δ_i - the delay directly due to the non-preemption
- $(q-1) \cdot L_{max}$ - the maximum network delay

Because the packet scheduling is not preemptive it can happen that a packet with high priority m_1 of any flow τ_i can be delayed by a lower priority packet m_2 of any flow belonging to lp_i if the high priority packet arrives at the processing node while the low priority packet is processed. In such a case, due to the non-preemption, the high priority packet needs to wait until the processing of the low priority packet is finished [MM05]. This the *direct* non-preemptive effect quantified by δ_i. We emphasize on *direct* because the non-preemptive effect has also an *indirect* component. In case packets belonging to hp_i would have arrived while m_1 was waiting for m_2, these packets, having a higher fixed priority than m_1, would have been processed before m_1. The delay that would have been incurred by m_1 in such a case represents the indirect non-preemptive effect. δ_i refers only to the maximum delay suffered by a packet of a flow τ_i *directly* due to flows belonging to lp_i. Upper bounds for δ_i are given in [MM05] (property 2) and in [MM06b] (property 1).

The processing time of a packet m of flow τ_i on node h we know it's expressed by C_i^h. It would be useful to have some notations with which upper bounds for the processing time of m along its trajectory could be expressed. Therefore, in [Mar04] the notation $slow_i$ representing the slowest node visited by τ_i on its trajectory is introduced. $slow_{j,i}$ represents the slowest node visited by flow τ_j on the path $\mathcal{P}_i$ followed by flow τ_i.

In order to obtain an upper bound for $W_i^q(t)$, an upper bound needs to be determined for $X_i(t)$. The non-preemptive effect is ignored in the analysis of $X_i(t)$. In the following we will briefly present the results given in [Mar04, MM05, MM06b]. Let's consider again the trajectory of packet m of flow τ_i that we

discussed in section 4.2.2.2. It is shown in [Mar04] (property 7.3.2) that only one packet is counted twice when considering two consecutive busy periods. That is, only one packet is common to two consecutive busy periods bp^h and bp^{h+1} and that packet is $f(h+1)$. In the worst-case, the packet that is counted twice is also the packet whose processing takes the longest [Bau11]. On a certain node h, the maximum processing time among all the packets visiting node h is denoted by $\max_{j \in hp_i \cup sp_i \cup \{i\}}\{C_j^h\}$.

Based on the previous result, we can identify two components of $X_i(t)$. The first one is represented by the packets that systematically appear in two consecutive busy periods and contribute to the packet m's delay. The second component is represented by the rest of the packets that can delay m. The contribution of each packet from the two categories is maximized by using in the expression of $X_i(t)$'s upper bound the longest processing time of the packet. The upper bound of $X_i(t)$ is given in Equation 4.12 [Bau11, Mar04].

$$X_i(t) \leq \sum_{g=f(1)}^{m} C_{\tau(g)}^{slow_{\tau(g),i}} - C_i^q + \sum_{h=1, h \neq slow_i}^{q} \max_{j \in hp_i \cup sp_i \cup \{i\}} \{C_j^h\} \tag{4.12}$$

An important question to be answered is which packets can belong to the busy periods of packet m of τ_i along its trajectory and, implicitly, contribute to $X_i(t)$? Which packets can delay m and how can their contribution to $X_i(t)$ be quantified? When should they be generated so that they can interfere with the transmission of packet m? We will get to the answers of these questions by analyzing the packets that can delay m separately, based on their membership to the sets hp_i, sp_i and $\{i\}$.

We start by analyzing a packet m' belonging to flow $\tau_j \in sp_i$. For the beginning, we consider only the case when flow τ_j and flow τ_i intersect in only one node h ($\mathcal{P}_i \cap \mathcal{P}_j = \{h\}$).

It is demonstrated in [Mar04] (propriety 7.3.3) that a packet belonging to a flow $\tau_j \in hp_i \cup sp_i \cup \{i\}$ can *not* delay m if (1) it arrives at the node $first_{i,j}$ before the packet $f(first_{i,j})$ or (2) it arrives at the node $last_{i,j}$ after the processing of m started. This implies, in our case, that in order for m' to delay m, m' must arrive at node h at soonest in the same time as $f(h)$ (whose arrival time is denoted by $a_{f(h)}^h$) and at latest in the same time as m.

So we need to find a lower bound for the (soonest) arrival time of $f(h)$ and an upper bound for the (latest) arrival of m at node h. The lower bound on the starting time of the busy period of flow τ_i at node h is denoted by M_i^h [Mar04, MM05], so we have $a_{f(h)}^h \geq M_i^h$. Because we are in the search of a lower bound we will consider the packets which have the shortest processing times and the

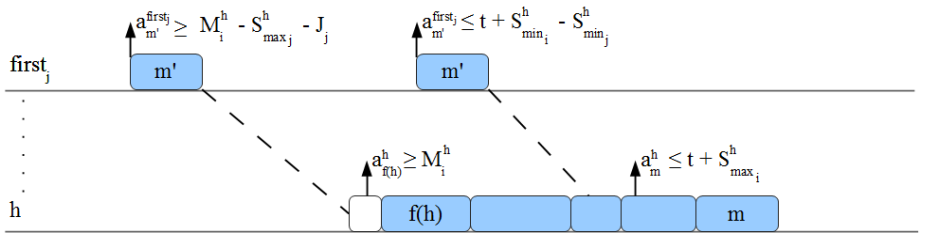


Figure 4.7: The interval of generation for a packet m' of flow $\tau_j \in sp_i$

shortest network delay L_{min}. This lower bound is obtained by summing up the shortest packet processing times and the shortest network delays on all the nodes of the τ_i's trajectory. The mathematical expression for M_i^h is given in Equation 4.13 [Mar04, Bau11], where the notation $pre_i(h)$ denotes the node visited by flow τ_i just before node h.

$$M_i^h = \sum_{h'=first_i}^{pre_i(h)} (\min_{j \in hp_i \cup sp_i \cup \{i\}, first_{i,j} = first_{j,i}} \{C_j^{h'}\} + L_{min}) \tag{4.13}$$

$S_{max_j}^h$ denotes the maximum time taken by a packet of flow τ_j to reach node h after being generated at its source node at time t (and, analogously, $S_{min_j}^h$ denotes the minimum time). We can see that $S_{max_i}^h$ is the upper bound we were searching for the arrival of m at node h.

We have now the upper and lower ends of the interval in which a packet m' of flow τ_j can arrive at node h and delay the processing of m. If $S_{max_i}^h$ is the last moment when m' could interfere in m's transmission, then in order for this interference to be possible m' must be generated at its source (node $first_j$) at latest $S_{min_j}^h$ earlier that $S_{max_i}^h$. Analogously, for m' to arrive at h in time to be comprised in the busy period, m' must be generated at soonest at $S_{max_j}^h - J_i$ before $a_{f(h)}^h$. Figure 4.7 [Bau11] illustrates the limits of the interval in which a packet m' of flow $\tau_j \in sp_i$ needs to be generated so it can delay m. In the figure, the white box represents a packet of lower priority than this of m, and the blue boxes represent packets with the same priority as this of m.

Let's consider now the more general case, when flow τ_j and τ_i don't intersect just in node h. Then the interval in which a packet of τ_j can be generated so it can delay a packet m of τ_i generated at time t has the duration $A_{i,j}$. The expression of $A_{i,j}$ is given in Equation 4.14 [MM06b, Bau11].

$$A_{i,j} = S_{max_i}^{first_{j,i}} - S_{min_j}^{first_{j,i}} + S_{max_j}^{first_{i,j}} - M_i^{first_{i,j}} + J_i \tag{4.14}$$

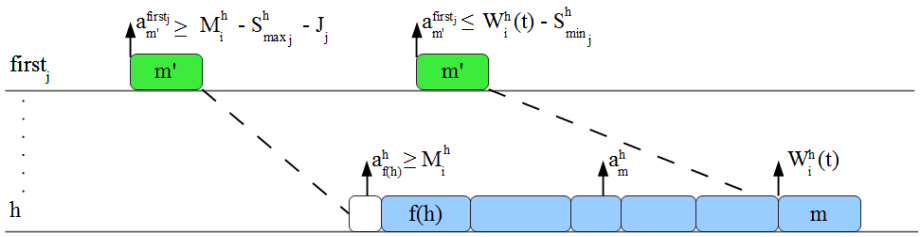


Figure 4.8: The interval of generation for a packet m' of flow $\tau_j \in hp_i$

We analyzed how packets of flows belonging to sp_i can delay m. In what is next we will focus on how packets of flows belonging to hp_i or other packets of flow τ_i can delay the packet under study.

Regarding the packets of flow τ_i, it is proven in [Mar04] that they can delay m (generated at time t and also a packet of τ_i) only if they are generated in the interval $[-J_i, t]$.

We will start the analysis of packet m' of flow $\tau_j \in hp_i$ with the simplifying assumption that τ_j and τ_i have only node h in common ($\mathcal{P}_i \cap \mathcal{P}_j = \{h\}$). We are again interested in the interval of generation of m' of τ_j so that it can delay m. The earliest m' can arrive at h and delay m is, as in the case of the flows belonging to sp_i, at the arrival of $f(h)$. The whole reasoning regarding the earliest generation time of m' that we previously have done for flows in sp_i is also valid for flows in hp_i. This is not the case for the latest generation time of m'.

Since τ_j has a higher fixed priority than this of τ_i, the latest arrival time of m' on node h so that it can delay m is limited by the start of the processing of m. So $W_i^h(t)$ constitutes the upper limit of the generation interval of m'. This interval is depicted in Figure 4.8 [Bau11], where with green boxes we represented the packets with higher static priority than this of m, with blue boxes we draw the packets having the same priority than m and with white boxes the packets with a lower priority than this of m.

If flows τ_j and τ_i have more than one node in common, then the last node on which m' could delay m is $last_{i,j}$. On this node the last time when m' could intervene in m's transmission is limited by $W_i^{last_{i,j}}(t)$. Considering the limit given by $W_i^{last_{i,j}}(t)$, the latest time of generation for m' on its source node $first_j$ so that it can arrive at $last_{i,j}$ in due time is $W_i^{last_{i,j}}(t) - S_{min_j}^{last_{i,j}}$.

We can conclude, as in [MM06b], that in order for packets of flow $\tau_j, j \in hp_i$

to delay packet m of flow τ_i generated at time t, the packets of τ_j need to be generated during the interval $[M_i^{first_{i,j}} - S_{max_j}^{first_{i,j}} - J_i, W_i^{last_{i,j}}(t) - S_{min_j}^{last_{i,j}}]$. The length of this interval is $W_i^{last_{i,j}}(t) + B_{i,j}$, where $B_{i,j}$ is given in Equation 4.15 [Bau11].

$$B_{i,j} = -S_{min_j}^{last_{i,j}} + S_{max_j}^{first_{i,j}} - M_i^{first_{i,j}} + J_i \tag{4.15}$$

We know that the maximum workload generated by any flow τ_j on node h in the interval $[a, b]$ is equal to $(1 + \lfloor (b-a)/T_j \rfloor)^+ \cdot C_j^h$ [MM06b], where $(1 + \lfloor a \rfloor)^+ = \max(0; 1 + \lfloor a \rfloor)$. Based on the previously obtained results regarding the contributions to $X_i(t)$ from all the categories of flows relative to τ_i (hp_i, sp_i and $\{i\}$), the following upper bound for $X_i(t)$ is obtained [MM06b, Bau11]:

$$\begin{aligned} X_i(t) \leq & \sum_{j \in hp_i} \left(\frac{W_i^{last_{i,j}}(t) + B_{i,j}}{T_j} \right)^+ \cdot C_j^{slow_{j,i}} + \sum_{j \in sp_i \cup \{i\}} \left(1 + \frac{t + A_{i,j}}{T_j} \right)^+ \cdot C_j^{slow_{j,i}} \\ & + \sum_{h \in \mathcal{P}_i, h \neq slow_i} \max_{j \in hp_i \cup sp_i \cup \{i\}} \{C_j^h\} - C_i^{last_i} \end{aligned} \tag{4.16}$$

The result from Equation 4.16 leads to the following upper bound for the starting time of packet m of flow τ_i on its last node [MM06b, Bau11]:

$$\begin{aligned} W_i^{last_i}(t) \leq & \sum_{j \in hp_i} \left(\frac{W_i^{last_{i,j}}(t) + B_{i,j}}{T_j} \right)^+ \cdot C_j^{slow_{j,i}} + \sum_{j \in sp_i \cup \{i\}} \left(1 + \frac{t + A_{i,j}}{T_j} \right)^+ \cdot C_j^{slow_{j,i}} \\ & + \sum_{h \in \mathcal{P}_i, h \neq slow_i} \max_{j \in hp_i \cup sp_i \cup \{i\}} \{C_j^h\} - C_i^{last_i} + \delta_i + (|\mathcal{P}_i| - 1) \cdot L_{max} \end{aligned} \tag{4.17}$$

This expression for $W_i^{last_i}(t)$ is to be used in Equation 4.11 to calculate the worst-case end-to-end response time of flow τ_i. The interval over which the maximization of R_i needs to be done is $t \geq -J_i$, so an upper bound for this interval would prove very useful. In [Mar04] a limit is given for this interval for scheduling policies such as FP/FIFO, FP/EDF or FIFO. Using this result, the expression of R_i over the restricted interval is given by Equation 4.18. Term $\mathcal{B}_i^{slow_i}$ is detailed in Equation 4.19 and verifies the relation $W_{i,t+\mathcal{B}_i^{slow_i}}^{last_i}(t) \leq W_{i,t}^{last_i}(t) + \mathcal{B}_i^{slow_i}$ for any time $t \geq -J_i$ [Mar04].

$$R_i = \max_{-J_i \leq t \leq -J_i + \mathcal{B}_i^{slow_i}} (W_{i,t}^{last_i} + C_i^{last_i} - t) \tag{4.18}$$

$$\mathcal{B}_i^{slow_i} = \sum_{j \in hp_i \cup sp_i \cup \{i\}} \left\lceil \frac{\mathcal{B}_i^{slow_i}}{T_j} \right\rceil \cdot C_j^{slow_{j,i}} \tag{4.19}$$

4.2.2.5 Applying trajectory approach to TTEthernet

In this section of the thesis, we will discuss the problematic of using the trajectory approach to determine worst-case end-to-end delays in TTEthernet networks. In which conditions can the trajectory approach be applied? How can the trajectory approach be extended for a complete applicability to TTEthernet networks? We mention from the start that we are only interested in the TT and RC traffic, since these are two traffic classes that can carry safety-critical messages and, consequently, timing guarantees are required for them.

A TTEthernet network is mapped to a trajectory approach distributed system in the same way as an AFDX configuration is:

- each TTEthernet switch output port (including the output link) corresponds to a processing node in the trajectory approach system
- the TTEthernet switching fabric represents a link in the trajectory approach system
- each virtual link path in a TTEthernet network defines a flow in the trajectory approach system

The TTEthernet RC traffic class is equivalent with the AFDX traffic. Therefore, it can be modeled in the trajectory approach in the same way as the AFDX traffic. More precisely, the trajectory approach flow τ_i corresponding to an RC frame f_i is defined by the following parameters (R represents the transmission rate for the TTEthernet switch output ports and output links):

$$J_i = 0,\ T_i = f_i.rate,\ C_i^h = f_i.size/R,\ D_i = f_i.deadline$$

While the mapping of the RC traffic to the trajectory approach concepts is straightforward, the mapping of the TT traffic class is more challenging. We identified two problems. One is that the TT frames are periodic and they have offsets and the trajectory approach doesn't consider offsets. The second one is that the TT frames can preempt the RC frames when the preemptive integration policy is employed in the TTEthernet network.

We thought about two possible ways in which the TT traffic could be handled by the trajectory approach. The first one implies that the trajectory approach is used only for the RC traffic. The TT static schedules would have been considered given (apriori known). In this case, the purpose of the analysis would have been to obtain the worst-case end-to-end delays of the RC frames taking into account the given TT schedules. This is a very similar problem to the one addressed by the TTEthernet analysis presented in section 4.1 and also by our TTEthernet simulator, which is described in section 4.3.

In conclusion, the trajectory approach needed to be extended to account for the TT static schedules. This way of framing the problem seemed promising because we already encountered a similar problem at the processor level. We described in Chapter 3 a schedulability analysis for FPS tasks when these tasks were allowed to execute only on their designated time partitions. Actually, the concepts of availability and demand, inspired by the approach in [PPEP08], that we used in our schedulability analysis, can be traced back to [PA00] where they were used in the timing analysis of the FTT-CAN protocol. Similarly to the partitioning scheme considered in Chapter 3, the TT static schedules would represent time partitions on the network level where the RC frames would not be allowed to be transmitted. In this context, to introduce the concepts of availability and demand (already successfully used both at processor and network level) in the trajectory approach seemed like a winning idea. However, due to the encountered difficulties we suspended the search for a solution in this direction. This was the first solution we envisioned for the problem of adapting the trajectory approach to consider the TT traffic. We will detail our second approach in the following.

The second way to integrate the TTEthernet TT traffic in the trajectory approach was based on the idea to model the TT frames as flows. For this integration, we already pointed out the problems of offsets and preemption. For the beginning, we make the simplifying assumption that the offsets are zero for all the TT frames on all the dataflow links and we treat the problem of preemption.

Since the TT traffic has priority over the RC traffic in the TTEthernet protocol, it is clear that the scheduling policy considered by the trajectory approach should be FP/FIFO.

In the trajectory approach, the scheduling of packets is considered to be non-preemptive. This means that once the transmission of a packet on a dataflow link was started, it can't be stopped. According to the shuffling integration policy in the TTEthernet protocol, the transmission of an RC frame once started it can not be interrupted by the arrival of TT frame. The TT frame is delayed until the transmission of the RC frame is finished. Based on these observations, we can say that the trajectory approach is a suitable timing analysis method

for TTEthernet networks where the TT and RC frames are integrated using shuffling.

But besides shuffling, there are two other integration policies for the RC and TT traffic that can be employed in a TTEthernet network: preemption and timely block. Timely block is not a preemptive strategy. On the contrary, it is based on the prevention of preemption. The decision of whether to send or not a RC frame is taken depending on the possible interference with the scheduled transmission of a TT frame. Both integration policies have some overheads: the timely block due to the computations that the switch needs to perform, the preemption policy due to the truncated RC frames which are sent in the network. If we consider these overheads equivalent, we saw in section 2.2.3 that the two integration policies have the same effect on the end-to-end response times. Therefore, we can conclude that from the point of view of an end-to-end delays analysis the two integration policies (timely block and preemption) are equivalent. Based on this result, from now on we will focus our attention on the TTEthernet networks where preemption is the employed integration policy.

We point out again that the trajectory approach employs a non-preemptive packet scheduling. We will try to extend it to consider a preemptive scheduling of packets so it can be applied to TTEthernet networks which use preemption as integration policy. A preemptive scheduling of packets means that a packet belonging to a flow with a higher static priority (a TT frame in our case) can preempt the "execution" of a packet of a flow with a lower static priority (a RC frame in our case). The transmission of a TTEthernet frame on a dataflow link is equivalent, in the trajectory approach model, with the execution of packet on a processing node. That processing node in the trajectory approach system corresponds to the TTEthernet switch output port and the dataflow link on which the packet is transmitted.

We think that even if we change one of the basic premises of the trajectory approach analysis (the non-preemptive packet scheduling), the principles and concepts behind trajectory approach remain valid. One thing that changes though, if we account for a preemptive packet scheduling, is the direct non-preemptive effect (denoted by δ in section 4.2.2.4). Practically, the direct and indirect non-preemptive effects disappear since the packets of flows with higher static priority can not be anymore delayed by packets of flows of lower static priority.

For assessing the impact of allowing packet preemption on the worst-case end-to-end delays, we go back to the study of a packet's trajectory, that we once did in section 4.2.2.4. Let's study the packet m belonging to flow τ_i. For the beginning, let's consider a packet m' belonging to flow $\tau_j \in sp_i \cup \{i\}$ with $\mathcal{P}_i \cap \mathcal{P}_j \neq \emptyset$. The contribution of packet m' to the worst-case end-to-end delay of m does

not change because the packet scheduling is now preemptive. According to the FP/FIFO scheduling algorithm that we consider in the network, packets like m' (that is, belonging to flows with a static priority as high as this of τ_i) can not preempt m. So, the way their contribution is calculated by the trajectory approach remains unchanged.

Let's consider now the case of packet m'' belonging to flow $\tau_j \in hp_i$ with $\mathcal{P}_i \cap \mathcal{P}_j \neq \emptyset$. We know that m'' can preempt m. The previous calculation of the contribution of the packets like m'' to the end-to-end delay of m was based on the premise that: "a packet of the flow τ_j can't affect m if it arrives after the moment when packet m has started execution on the last node $last_{i,j}$". This implies, as shown in section 4.2.2.4, that the packet m'' needs to be generated in the interval $[M_i^{first_{i,j}} - S_{max_j}^{first_{i,j}} - J_i, W_i^{last_{i,j}}(t) - S_{min_j}^{last_{i,j}}]$.

Since preemption is allowed, the last moment when the packet m'' of τ_j can be generated and affect the end-to-end delay of packet m needs to be extended with the processing time of packet m on node $last_{i,j}$. We can say that we are not interested anymore in the latest starting time of packet m on $last_{i,j}$, but on the latest termination time on this node. The new limits for the interval are then $[M_i^{first_{i,j}} - S_{max_j}^{first_{i,j}} - J_i, W_i^{last_{i,j}}(t) - S_{min_j}^{last_{i,j}} + C_i^{last_{i,j}}]$.

We propagate this change through the subsequent calculations and we obtain the following expression for $W_i^{last_i}(t)$, valid for a preemptive trajectory approach:

$$\begin{aligned} W_i^{last_i}(t) \leq & \sum_{j \in hp_i} \left(\frac{W_i^{last_{i,j}}(t) + B_{i,j} + C_i^{last_{i,j}}}{T_j} \right)^+ \cdot C_j^{slow_{j,i}} \\ & + \sum_{j \in sp_i \cup \{i\}} \left(1 + \frac{t + A_{i,j}}{T_j} \right)^+ \cdot C_j^{slow_{j,i}} \\ & + \sum_{h \in \mathcal{P}_i, h \neq slow_i} \max_{j \in hp_i \cup sp_i \cup \{i\}} \{C_j^h\} - C_i^{last_i} + (|\mathcal{P}_i| - 1) \cdot L_{max} \end{aligned} \tag{4.20}$$

This expression for $W_i^{last_i}(t)$ from Equation 4.20 can be then inserted in Equation 4.11 to calculate the worst-case end-to-end delay of flow τ_i.

Summarizing the findings from this section, we can say that the trajectory is partially suitable to be used in the timing analysis of TTEthernet. We saw that the application of trajectory approach to TTEthernet networks is not straightforward. Simplifying premises (like not considering offsets in the TT traffic modeling) are needed. The integration policy of the TT and the RC traffic that is employed has also a say in how the trajectory approach can be applied to TTEthernet. We have shown that, in case shuffling is used, the "original" tra-

jectory approach with a FP/FIFO non-preemptive scheduling policy works well for TTEthernet networks. In order for the trajectory approach to be applicable to TTEthernet when the other integration policies are employed, we needed to challenge the non-preemptive packet scheduling premise. We proposed an extension of the trajectory approach that caters for a preemptive scheduling of the packets in the network.

4.3 A TTEthernet Simulator

We implemented a simulator for a TTEthernet network with the purpose of determining the end-to-end delays experienced by the RC frames. A high-level view of the simulator is provided by Figure 4.9. As it can be seen from Figure 4.9, the inputs of the simulator are:

- the network topology - that is, the layout of the TTEthernet cluster that we are simulating: the end systems and the switches and how they are connected by physical links.
- the virtual links describing the communication paths in the network
- the RC and TT messages to be sent over the network, specified by their parameters (size, period for TT messages and rate for RC messages, deadline) as it was described in section 2.2.4
- the assignment of messages to virtual links
- the TT schedule tables - the communication schedules for the defined TT messages containing the sending and receiving times on all the dataflow links during one application cycle

We consider the TTEthernet to be composed of a single TTEthernet cluster, so we have a single clock synchronization domain. We assume that there is no BE traffic on the network, that one message is carried by one frame and that there are no errors, packet losses or link failures. The TT traffic is fully described by the simulator's inputs, so our simulator is concerned only with the analysis and simulation of the RC traffic. A more complete simulator for TTEthernet, constructed on the basis of the OMNeT++ INET framework, is described in [SKKS11].

As mentioned, the output of the simulator is represented by the worst-case values for the end-to-end delays of the RC frames that were encountered during

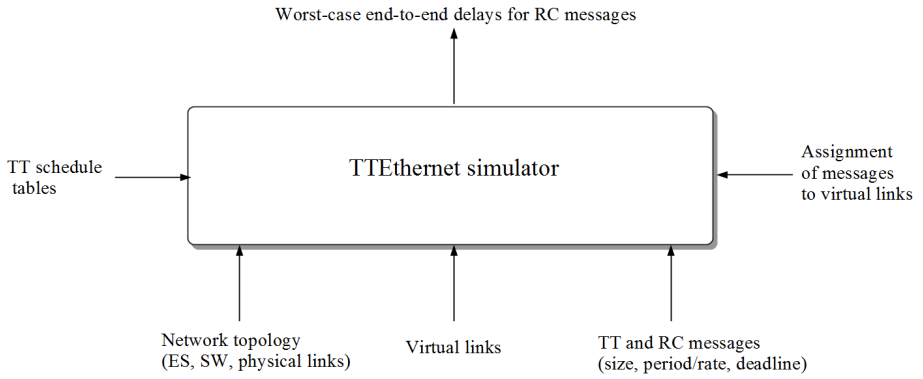


Figure 4.9: TTEthernet simulator

the simulation. Based on the determined worst-case end-to-end delays we can assess the schedulability of the rate-constrained frames. Since what we are doing is a simulation, if we find that the deadline is respected for a RC frame, we can not certainly say that the RC frame is schedulable. But in case we find an end-to-end delay greater than the deadline, we can say with certainty that the RC frame is not schedulable.

The simulator calculates the end-to-end delays for the RC frames in the way that was shown in [TSP12]. The worst-case end-to-end delay of an RC frame can be viewed as having the following two components:

- the queuing delays experienced by the RC frame in the network nodes due to TT frames or other RC frames being serviced before
- the network delays (basically, the durations of the RC frame's transmissions on the dataflow links in the network)

This decomposition is expressed more formally in Equation 4.21, using the notations presented in section 2.2.4. We denote by R_{f_i} the worst-case end-to-end delay of a RC frame $f_i \in \mathcal{F}^{RC}$ sent on the virtual link vl_i.

$$R_{f_i} = \sum_{v_j, v_k \in V, [v_j, v_k] \in vl_i} (Q_{f_i}^{[v_j, v_k]} + C_{f_i}^{[v_j, v_k]}) \tag{4.21}$$

The queuing delay suffered by frame f_i in the network node $v_j \in \mathcal{V}$, while waiting to be transmitted on the dataflow link $[v_j, v_k]$ is further decomposed based on its causes (Equation 4.22) [TSP12].

$$Q_{f_i}^{[v_j, v_k]} = Q_{f_i, [v_j, v_k]}^{TT} + Q_{f_i, [v_j, v_k]}^{RC} + Q_{v_j}^{TL} \tag{4.22}$$

The delay induced by the TT traffic happening from the moment of f_i's arrival at the node v_j until f_i is sent to the node v_k is denoted by $Q^{TT}_{f_i,[v_j,v_k]}$. f_i is also delayed by the RC frames that arrived before it at the node and, according to the TTEthernet FIFO policy for RC frames, are serviced before f_i. This delay is expressed by $Q^{RC}_{f_i,[v_j,v_k]}$. The last part of $Q^{[v_j,v_k]}_{f_i}$, $Q^{TL}_{v_j}$, refers to the so-called technical latency that is induced by the hardware tasks implementing the TTEthernet protocol on the network node [TSP12].

We will exemplify the calculation of the end-to-end delay for an RC frame, using the transmission scenario in Figure 4.10. The network topology underlying the example is the one that was discussed in section 2.2.5 and presented in Figure 2.9. We can see that there are four frames implicated in the scenario. f_1 and f_2 are being sent from ES_1 to ES_2 and need to cross switch SW_1 in their way. f_3 and f_4 are originating from SW_2 and SW_3, respectively, and they also cross SW_1 in their way to ES_2. f_1 and $f_3 \in \mathcal{F}^{RC}$, while f_4 and $f_2 \in \mathcal{F}^{TT}$. In our analysis, we are concerned with the end-to-end delay of frame f_1. We consider that all the physical links have the same speed. This implies that a frame's transmission duration is the same on all dataflow links in the network. For the four frames in our scenario, the transmission durations are as follows: for f_1 - $C_1 = 120\ \mu s$, for f_2 - $C_2 = 125\ \mu s$, for f_3 - $C_3 = 80\ \mu s$ and for f_4 - $C_4 = 100\ \mu s$. The method employed in our example for the integration of the TT and RC traffic is preemption. Our example is very similar (we use the same set of frames and the same transmission parameters) to an example in [TSP12], with the difference that in [TSP12] the considered integration policy is timely block.

In Figure 4.10 we have illustrated the scenario that leads to the worst-case end-to-end delay (denoted R_{f_1}) for the frame under analysis, f_1. This situation occurs for the frame instance $f_{1,1}$, when f_1 is delayed by all the other frames in the system (namely, the frame instances $f_{2,1}$, $f_{3,1}$ and $f_{4,1}$). We assume that, according to the TT static schedule for the dataflow link $[SW_1, ES_2]$, the TT frames $f_{2,1}$ and $f_{4,1}$ are scheduled for transmission at 130 μs and 310 μs, respectively. Since the transmission duration of the frame instance $f_{2,1}$ is $C_2 = 125\ \mu s$, the dataflow link $[SW_1, ES_2]$ will be free in the interval (255 μs, 310 μs). In these conditions, it would be disadvantageous for $f_{1,1}$ to arrive at SW_1 at the 250 μs. Because the frame instance $f_{4,1}$ is scheduled at 310 μs, $f_{1,1}$ would not have enough time for a complete transmission. However, since preemption is employed in the network, $f_{1,1}$ would be transmitted by the switch anyway and its transmission would be interrupted when $f_{4,1}$ arrives. Since the RC frames can arrive at arbitrary times, we will consider this moment of arrival for $f_{1,1}$ (250 μs) when building the worst-case end-to-end delay scenario for $f_{1,1}$. The preempted, incomplete transmission of $f_{1,1}$ together with the technical latency, $Q^{TL}_{v_j}$, need to be included in the queuing delay due to TT frames. For

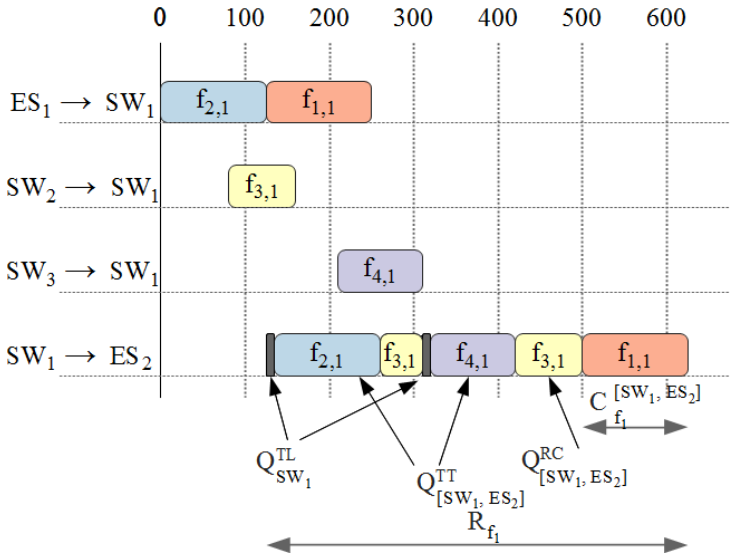


Figure 4.10: End-to-end delay analysis for frame f_1

the technical latency we will use in our example a value of 5 μs. We can then evaluate the delay caused by the TT traffic as being $Q^{TT}_{f_1,[SW_1,ES_2]} = 285\ \mu s$.

So far, in our analysis, we considered only the TT interference. We also need to consider the RC interference in f_1's worst-case end-to-end delay. Obviously, for our case, this means that the RC frame instance $f_{3,1}$ arrives at switch SW_1 before $f_{1,1}$. Because the frames belonging to the same traffic class are treated in FIFO order, this phasing of the frames would introduce a supplementary delay for $f_{1,1}$: $Q^{RC}_{f_{1,1},[SW_1,ES_2]} = C^{[SW_1,ES_2]}_{f_3} = C_3 = 80\ \mu s$. As it is depicted in Figure 4.10, since $f_{3,1}$ arrives before $f_{1,1}$ at SW_1 and preemption is in use, $f_{3,1}$ will be the preempted frame instead of $f_{1,1}$.

Summing up, the delay suffered by $f_{1,1}$ in the switch SW_1 is $Q^{[SW_1,ES_2]}_{f_{1,1}} = Q^{TT}_{f_{1,1},[SW_1,ES_2]} + Q^{RC}_{f_{1,1},[SW_1,ES_2]} = 365\ \mu s$. In this case, the worst-case end-to-end delay of f_1 is $R_{f_1} = C^{[ES_1,SW_1]}_{f_1} + Q^{[SW_1,ES_2]}_{f_{1,1}} + C^{[SW_1,ES_2]}_{f_1} = 605\ \mu s$.

4.3.1 Implementation

As we previously stated in the beginning of this section, the RC traffic is simulated on the basis of a given network architecture and known TT static schedules.

The simulator inputs (as well as the outputs) are stored in files. When starting a simulation, the following files are read:

- `network.graphml` - the GraphML [1] file that contains the network's topology (the end systems, the switches and how they are connected)
- `vls` - the file containing the virtual link's dataflow paths
- `msg` - the file where the size, the deadline, the period or the rate of the TT and RC messages are specified. This file also contains the mapping of messages to virtual links.
- `historyOPT` - the file that contains the TT static schedules for all the dataflow links in the network. The TT schedule tables contained in this file are the result of the Tabu Search-based optimization strategy for the synthesis of TT schedules proposed in [TSP12].

One may say that since we have the TT static schedules and we only simulate the RC traffic in the network, to have the TT messages also specified in the file *msg* is redundant or useless. We preferred to keep it like this for the sake of the simulator's extensibility. The simulator is designed in such a way that, after a minimum implementation effort, it should be able to simulate the TT traffic in the network as well. Besides this point, all the simulator's input files (with the exception of *historyOPT*) served as input configuration for the TT schedule tables synthesis tool developed in [TSP12]. Practically, we reuse the input and output files of [TSP12], our tool having a different destination.

The simulator was implemented in Java, in about 40 source files grouped in three packages. One of the packages is `tte.network` which contains the data structures used to represent the TTEthernet network in our program. The correspondence between our data structures and the network elements they model is pretty straightforward as it can be seen from Figure 4.11. A `Network` object aggregates `NetworkNode` objects (that is, either `EndSystem`, either `Switch` objects) and `DataflowLink` objects. Since the network topology is given in a GraphML file, so it is already specified as a graph in the input file, it was natural to keep this graph representation for the `Network` objects. We used for this the JUNG (Java Universal Network/Graph) Framework [2] and so the `Network` class extends the `DirectedSparseMultigraph` JUNG class. We note here that the `VirtualLink` class, which models the virtual links in our program, is also a JUNG `DirectedSparseMultigraph` (its vertices and edges are the IDs of the network elements found on the virtual link's dataflow paths). So, a virtual link is actually a subgraph of the network graph.

[1] GraphML - an XML-based format for representing graphs
[2] JUNG - a Java open-source library for graph modelling and visualization

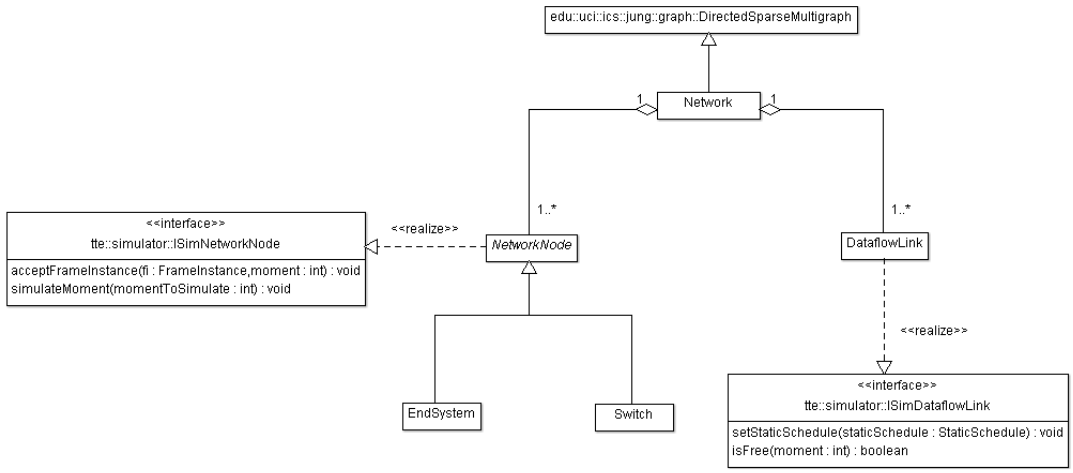


Figure 4.11: The network model

The behavior of the end systems and switches during simulation is described by the implementation of the interface `ISimNetworkNode` in the classes `EndSystem` and `Switch`. The method `acceptFrameInstance()` is called whenever a frame reaches a network element, either coming from another network element on a dataflow link, either, in the case of end systems sources of virtual links, after just being generated. The method `simulateMoment()` is called for every simulation step, the parameter of the method telling the network element the current simulation time. The class `DataflowLink` implements the interface `ISimDataflowLink`. By calling the methods defined in the interface one may assign a certain TT static schedule to the dataflow link (the static schedules are modelled by the `StaticSchedule` class) or check if the dataflow link is carrying any TT or RC traffic at a given time.

We are simulating a single TTEthernet cluster, so we have a single clock synchronization domain, that is, all the network elements have the same sense of time. In this context, we preferred to implement a time driven simulator (instead of an event driven one). The core of the simulator is the `Simulator` class from the package `tte.simulator`. This class keeps track of the simulation time and of the simulated messages, and drives the simulation. Since our assumption is that every messages fits in exactly one frame, we use the class `Frame` to represent a message read from the `msg` input file. A `Frame` object is a simple container for the parameters (size, deadline, message ID, rate, the ID of the virtual link to which the message is assigned to) of the corresponding message. The object also

stores the scenario that lead to the obtained worst-case end-to-end delay for the frame (this scenario is kept in a `FrameInstTransmissionScenario` object). In our implementation, a frame instance is not represented by an instance of the `Frame` class, but by an instance of the `FrameInstance` class. A `FrameInstance` object keeps a reference to its corresponding `Frame` object. A simplified UML diagram corresponding to the simulator's implementation is provided in Figure 4.12.

The frame instances to be simulated are created during the initialization phase of the simulation. For each frame instance we will calculate the end-to-end delay. This delay is the temporal distance between the moment in which the frame instance is sent to the communication services of the end system by an application task (the so-called release time) and the moment in which the frame instance is available for reading by the destination task at the destination end system (the so-called arrival time). The asynchronous nature of the RC traffic is simulated by assigning to the frame instances random arrival times at their source end systems.

The most relevant aspects of the simulation routine are presented in Algorithm 2. As we previously mentioned, the simulator first reads the input files creating the object representation of the input configuration. The simulator is initialized with the number of application cycles to be simulated and, based on this, calculates the number of simulation steps (line 13 in Algorithm 2). Once the RC frame instances to be simulated are created and initialized with random release times (line 14), the simulation loop can begin. In each simulation step, the frame instances which are released in that moment are collected (line 17) and "placed" to their corresponding source end systems (lines 18-21). By this placement of the frame to its source end system, the simulator emulates the behavior of a task that, in real life, would send a message to the end system's communication ports. This corresponds to step 1 from the RC frames' transmission in a TTEthernet network that we described in section 2.2.5. After introducing the newly released frames in the scene, the simulator gives the control to the network nodes (the end systems and the switches) (lines 22-24). In their implementation of the method `simulateMoment()` from the interface `ISimNetworkNode`, the end systems and the switches handle the frames found in their internal buffers and queues, and start the frames' transmission on the dataflow links when it's the case. In our program, the assumption is that the queues and buffers in switches and end systems are large enough to accommodate worst-case traffic. The traffic shaping and the traffic policing roles of the end systems and, respectively, of the switches are being implemented in the method `simulateMoment()`.

Like in the real world, in our implementation, the dataflow link is a passive element that does not "move" frames, but is just transited by frames. The frames are placed on the dataflow links by the network nodes (end systems and

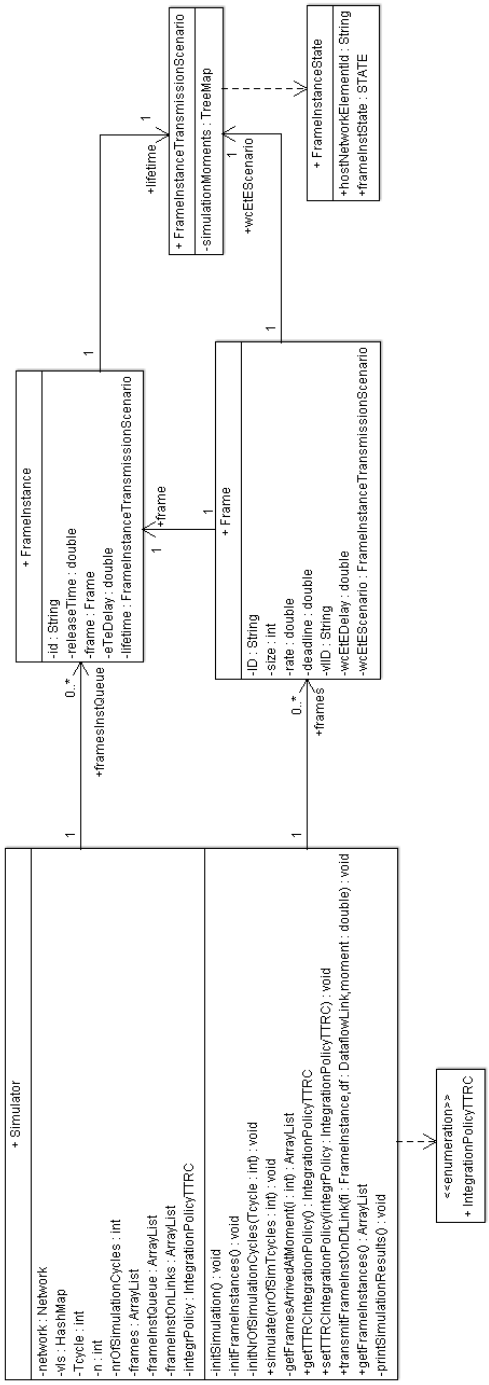


Figure 4.12: Simplified UML diagram of the simulator implementation

Algorithm 2 CENTRAL_SIMULATION_ALGORITHM

Inputs:
$\mathcal{G}(\mathcal{V}, \mathcal{E})$ - the network topology;
$\mathcal{F}^{RC}$ - the set of RC messages to simulate;
$\mathcal{VL}$ - the set of virtual links;
$\mathcal{M}$ - the assignment of frames to virtual links;
T_{cycle} - the application cycle;
$\mathcal{S}$ - the set of TT static schedules;
n - the number of application cycles to simulate;
Outputs:
$WCETED$ - the worst-case end-to-end delay for each frame $f \in \mathcal{F}^{RC}$

begin
$simCycles \leftarrow n \times T_{cycle}$
$frameInstQueue \leftarrow InitFrameInstances(\mathcal{F})$
$cycle \leftarrow 0$
repeat
  $arrivedFrameInst \leftarrow GetArrivedFrames(frameInstQueue, cycle)$
  for all $f_i \in arrivedFrameInst$ **do**
    $es \leftarrow GetSourceEndSystem(f_i)$
    $es.acceptFrameInstance(f_i)$
  end for
  for all $networkNode \in \mathcal{V}$ **do**
    $networkNode.simulateMoment(cycle)$
  end for
  $frameInstOnLinks \leftarrow GetFramesTransmittedOnLinks(cycle)$
  for all $f_i \in frameInstOnLinks$ **do**
    if $IsFrameTransmissionOnLinkDone(f_i)$ **then**
      if $HasFrameReachedFinalDestination(f_i)$ **then**
        $f_i.eToEDelay \leftarrow cycle - f_i.start$
        $ComputeWCETEDelay(f_i.frame, f_i.eToEDelay)$
      else
        $networkNode \leftarrow GetNextNetworkNodeOnPath(f_i)$
        $networkNode.acceptFrameInstance(f_i)$
      end if
    end if
  end for
  $cycle++$
until $cycle = simCycles$
end

switches) they crossed in their path. The simulator keeps track of the frames

transiting dataflow links in each simulation step (lines 25-36 in Algorithm 2). The simulator assesses if the transmission of each of these frames on the corresponding dataflow link is ending in the current simulation step (line 27). In case it does, the frame needs to be placed in the next network element on its path. When the frame reaches its final destination, the simulator computes the end-to-end delay experienced by the frame during its transmission (line 29). This delay will be stored if it is the maximum delay suffered by the frame so far in the simulation (line 30).

4.3.2 Testing

The testing of our program was done using unit tests, as well as pencil and paper tests. The unit tests were a quick and effective way to detect problems in the way we read the input configuration. Besides validating the way we handled the inputs, we also wrote unit tests for the important methods in the simulation algorithm. The implementation of the unit tests is to be found in the packages `tte.test.*`. To test our program, not just with automated tests, but also by using pencil and paper, proved to be necessary and very useful. Using simple input configurations for which we could manually calculate the worst-case end-to-end delays, we checked that our algorithm gives the same results. In the cases when it didn't, the debugging of the program was also made simpler by using small input configurations.

4.3.3 Evaluation

It is of vital importance that the results of a simulation are evaluated by comparing them either with real-life measurements or with results provided by a formal framework. In our case, the evaluation of the simulation results was done against the results obtained by running the TTEthernet analysis presented in section 4.1. We have used 11 synthetic benchmarks which were derived from the results obtained by the static schedules' synthesis optimization from [TSP12]. As we previously mentioned, the input static schedules used by the simulator and by the TTEthernet analysis are the optimized static schedules obtained with the approach from [TSP12]. The simulations were run on a Dell laptop equipped with a Intel T6670 CPU and 4 GB RAM. Depending on the length of the application cycle and on the number of application cycles to be simulated, one simulation took from 4-5 minutes to 40-50 minutes.

The experimental results we obtained are presented in Table 4.2. We simulated 5, 10 and 20 application cycles for each benchmark. In the table, we display

Table 4.2: Experimental results

Test Case	ES	SW	RC messages	Frame instances	$\Delta_{e-t-e\ delay}$ [%]
1	11	4	40	5200	21.60
2	13	3	30	3900	66.10
3	25	6	30	3600	24.76
4	25	6	70	1400	33.41
5	25	6	60	1200	29.45
6	35	8	20	4000	57.85
7	35	8	40	1680	46.44
8	35	8	60	3780	27.49
9	35	8	70	4410	25.48
10	35	8	60	1260	27.56
11	45	10	60	3780	56.13

only the best simulation result for each benchmark. The number of end systems and switches composing the TTEthernet network is presented in columns 2 and, respectively, 3. Column 4 contains the number of rate-constrained messages simulated and column 5 contains the total number of frame instances that were released during the simulation. The last column in Table 4.2 presents the percentage difference between the worst-case end-to-end delays obtained by the simulator and by the analysis averaged over all messages.

Chapter 5

Conclusions

The safety-critical systems need to have deterministic behaviors, since their hazardous functioning can endanger life or the environment. The mixed-criticality systems, which have safety-critical functions among other non-critical ones, need to guarantee that their safety-critical functions are operating hazard-free. This is one reason why analysis tools that can determine worst-case response times for tasks and worst-case end-to-end delays for messages are needed.

The subject of this thesis was the analysis of mixed-criticality applications implemented on distributed architectures. In order to have functions with different criticalities sharing the same computing platform, one must ensure that there is enough spatial and temporal separation among them. This is also true for the communication level, when messages with different criticalities are transmitted on the same network. Our thesis addressed the problematic of mixed-criticality applications on distributed partitioned architectures both at platform level, as well as communication level.

IMA analysis. For the separation required at the level of the processing nodes (elements) we considered an IMA-like partitioning scheme. Applications were allowed to execute only on their designated partitions (that is, designated time slots on the processor) and each partition could define its own scheduling policy. We were interested only in the timing analysis of the applications running in partitions that employ fixed-priority preemptive scheduling (FPS). The analysis

we proposed for these applications was based on the WCDOPS+ algorithm. We have extended the algorithm to consider a non-periodic partitioning scheme. We have then compared the worst-case response times that we obtained from running our algorithm to the results given by the state-of-the-art analysis for FPS tasks executed on partitioned architectures proposed by Audsley and Wellings. Our algorithm gave less pessimistic results and also has the advantage that it can consider non-periodic partitions. As previously mentioned, based on this work, we wrote a paper presented in the work in progress section of the ETFA 2012 conference.

TTEthernet analysis and simulation. For the separation required at the communication level we have chosen the TTEthernet protocol. The separation between mixed-criticality messages is achieved in a TTEthernet network through the concept of virtual links. TTEthernet integrates three traffic classes: TT (time-triggered) messages which are transmitted according to static schedule tables, RC (rate-constrained) messages which are asynchronous messages with bounded end-to-end delay and BE (best effort) messages for which there are no timing guarantees provided. In this thesis we were concerned only with the end-to-end delay analysis of RC messages. We have extensively studied the trajectory approach analysis method for end-to-end delays in AFDX networks and proposed an extension of it in order to enable it to be applied in TTEthernet networks as well. We have built a TTEthernet simulator which determines the worst-case end-to-end delay of RC messages. The simulation results were compared with the end-to-end delays obtained with the previously proposed TTEthernet analysis.

5.1 Future Work

It is our belief that the relatively young field of mixed-criticality systems offers broad research perspectives. For example, the flexibility and extensibility of embedded systems' architectures are considered to be top research priorities by the ARTEMIS Research Agenda [ART11]. Also, the optimization of mixed-criticality applications implemented on distributed architectures - our thesis dealt only with the analysis - is a promising direction in which there is ongoing research work [TSP11b, TSP11a, TSMP12].

From a closer perspective to our thesis, we list here some suggestions for improvements and future work:

- the TTEthernet simulator could be improved by taking into consideration the BE traffic. The simulator could also consider packet losses or link

failures.

- the TTEthernet networks that we considered both for analysis and simulation were composed of a single cluster. Multi-cluster networks would be interesting to analyze, since the TTEthernet messages can change their traffic classes when moving from one cluster to another.
- the IMA analysis we proposed did not consider communication. A holistic timing analysis that would integrate our analyses done at the CPU-level and at the communication level would be a valuable improvement for our work.
- the analysis of heterogeneous real-time networks is also interesting. There is ongoing work in the analysis using trajectory approach of network architectures consisting of CAN buses interconnected by a AFDX. network [LSF12]. Using TTEthernet in such architectures is an option worth considering.
- the holistic analysis method for AFDX-based distributed systems proposed in [GPGH11, GPGH12] could be extended for TTEthernet-based distributed architectures.
- a study of the influence that different frame parameters (like, for example, the frame rate or period) have on the frames' end-to-end delays in TTEthernet networks could be made.
- we considered the topology of the virtual links given. Various optimization problems can be defined related to the design of the virtual links. For example, one could determine the virtual links' topology for which the end-to-end delays are minimized.
- it would be interesting to integrate the tools developed for this project into existing embedded systems toolchains.

Bibliography

[Ari91] ARINC specification 651: Design guidance for integrated modular avionics. Technical report, Aeronautical Radio Inc., Annapolis, USA, 1991.

[Ari93] ARINC specification 659: Backplane data bus. Technical report, Aeronautical Radio Inc., Annapolis, USA, 1993.

[Ari96] ARINC specification 629: Multi-transmitter data bus; part 1, technical description (with five supplements); part 2, application guide (with one supplement). Technical report, Aeronautical Radio Inc., Annapolis, USA, 1996.

[Ari97] ARINC specification 653: Avionics application software standard interface. Technical report, Aeronautical Radio Inc., Annapolis, USA, 1997.

[ARI09] ARINC Report 664P7-1. *Aircraft Data Network, Part 7: Avionics Full Duplex Switched Ethernet (AFDX) Network*, September 2009.

[ART11] ARTEMIS Office, ARTEMIS programme. Strategic research agenda, 2011.

[as611] *AS6802: Time-Triggered Ethernet.* SAE International, 2011.

[ASF11] Muhammad Adnan, Jean-Luc Scharbarg, and Christian Fraboul. Minimizing the search space for computing exact worst-case delays of afdx periodic flows. In *SIES*, pages 294–301, 2011.

[ATB93] N. Audsley, K. Tindell, and A. Burns. The end of the line for static cyclic scheduling. In *Proceedings of Euromicro Workshop on Real-Time Systems*, 1993.

[AW96] N. Audsley and A. Wellings. Analysing APEX applications. In *Real-Time Systems Symp.*, pages 39 –44, 1996.

[Bau11] Henri Bauer. *Analyse pire cas de flux hétérogenes dans un réseau embarqué avion*. These, Université de Toulouse, October 2011.

[BBB+09] James Barhorst, Todd Belote, Pam Binns, John Hoffman, James Paunicka, Prakash Sarathy, John Scoredos, Peter Stanfill, Douglas Stuart, and Russell Urzi. White paper: A research agenda for mixed-criticality systems, 2009.

[BHR90] Sanjoy K. Baruah, Rodney R. Howell, and Louis Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2:301–324, 1990.

[BSF09] H. Bauer, J.-L. Scharbarg, and C. Fraboul. Applying and optimizing trajectory approach for performance evaluation of AFDX avionics network. In *Proceedings of the IEEE Conference on Emerging Technologies Factory Automation*, 2009.

[BSF12] Henri Bauer, Jean-Luc Scharbarg, and Christian Fraboul. Applying trajectory approach with static priority queuing for improving the use of available afdx resources. *Real-Time Systems*, 48(1):101–133, 2012.

[Car04] Larry Carley. Brake-By-Wire, 2004.

[CC93] P. Chanet and V. Cassigneul. How to control the increase in the complexity of civil aircraft onboard systems, 1993.

[Cru91] Rene L. Cruz. A calculus for network delay, part I & II. *IEEE Transactions on Information Theory*, 37(1), 1991.

[CSEF06] Hussein Charara, Jean-Luc Scharbarg, Jerome Ermont, and Christian Fraboul. Methods for bounding end-to-end delays on an afdx network. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems*, ECRTS '06, pages 193–202, Washington, DC, USA, 2006. IEEE Computer Society.

[Dic] Robert Dick. Embedded system synthesis benchmarks suite. http://ziyang.eecs.umich.edu/~dickrp/e3s/.

[FFG06] Fabrice Frances, Christian Fraboul, and Jérôme Grieu. Using network calculus to optimize the AFDX network. In *European Congress on Embedded Real-Time Software (ERTS), Toulouse France, 25/01/06-27/01/06*, page (electronic medium). SIA/3AF/SEE, 2006.

[Fid98] CJ Fidge. Real-time schedulability tests for preemptive multitasking. *REAL-TIME SYSTEMS*, 14(1):61–93, 1998.

[Foh94] G. Fohler. Flexibility in statically scheduled hard real-time systems, 1994.

[GFF03] Jérome Grieu, Fabrice Frances, and Christian Fraboul. Preuve de determinisme d'un reseau embarque avionique . In *Colloque Francophone sur l'ingénierie des protocoles 2003 , Paris, 07/10/2003-10/10/2003*, page 16. Hermès, octobre 2003. (Conférencier invité).

[GPGH11] J. J. Gutiérrez, J. C. Palencia, and M. González Harbour. Response time analysis in AFDX networks, 2011.

[GPGH12] J. J. Gutiérrez, J. C. Palencia, and M. González Harbour. Response time analysis in AFDX networks with sub-virtual links and prioritized switches, 2012.

[HD93] K. Hoyme and K. Driscoll. SAFEbus. *IEEE Aerospace Electronic Systems Magazine*, 8, 1993.

[KAGS05] Hermann Kopetz, Astrit Ademaj, Petr Grillinger, and Klaus Steinhammer. The time-triggered ethernet (tte) design. In *Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, ISORC '05, pages 22–33, Washington, DC, USA, 2005. IEEE Computer Society.

[KM07] Jarik P. Kany and Sigurd H. Madsen. Design optimisation of fault-tolerant event-triggered embedded systems. Master's thesis, Dept. of Informatics and Mathematical Modelling, Technical University of Denmark, 2007.

[Kop11] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer Verlag, 2nd edition, 2011.

[LBT01] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer-Verlag, Berlin, Heidelberg, 2001.

[LSF10] Xiaoting Li, Jean-Luc Scharbarg, and Christian Fraboul. Improving end-to-end delay upper bounds on an afdx network by integrating offsets in worst-case analysis. In *ETFA*, pages 1–8, 2010.

[LSF12] Xiaoting Li, Jean-Luc Scharbarg, and Christian Fraboul. Applying trajectory approach for worst-case delay analysis of a heterogeneous real-time network. In *Proceedings of the 18th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2012.

[Mar04] Steven Martin. *Maîtrise de la dimension temporelle de la qualité de service dans les réseaux*. These, Université Paris XII Val de Marne, July 2004.

[MM05] Steven Martin and Pascale Minet. Improving the analysis of distributed non-preemptive fp/dp* scheduling with the trajectory approach. *Telecommunication Systems*, 30:49–79, 2005. 10.1007/s11235-005-4315-2.

[MM06a] Steven Martin and Pascale Minet. Schedulability analysis of flows scheduled with fifo: application to the expedited forwarding class. In *Proceedings of the 20th international conference on Parallel and distributed processing*, IPDPS'06, pages 180–180, Washington, DC, USA, 2006. IEEE Computer Society.

[MM06b] Steven Martin and Pascale Minet. Worst case end-to-end response times of flows scheduled with fp/fifo. In *Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies*, ICNICONSMCL '06, pages 54–, Washington, DC, USA, 2006. IEEE Computer Society.

[MTSAP12] Sorin Ovidiu Marinescu, Domiţian Tămaş-Selicean, Vlad Acretoaie, and Paul Pop. Timing analysis of mixed-criticality hard real-time applications implemented on distributed partitioned architectures. In *Proceedings of the 17th IEEE Conference on Emerging Technologies and Factory Automation*, 2012.

[PA00] P. Pedreiras and L. Almeida. Combining event-triggered and time-triggered traffic in FTT-CAN: analysis of the asynchronous messaging system. In *Factory Communication Systems, 2000. Proceedings. 2000 IEEE International Workshop on*, pages 67 –75, 2000.

[PEP05] P. Pop, P. Eles, and Z. Peng. Schedulability-driven frame packing for multicluster distributed embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 4(1):112–140, 2005.

[PPE+08] Traian Pop, Paul Pop, Petru Eles, Zebo Peng, and Alexandru Andrei. Timing analysis of the FlexRay communication protocol. *Real-Time Systems*, 39(1-3):205–235, 2008.

[PPEP08] Traian Pop, Paul Pop, Petru Eles, and Zebo Peng. Analysis and optimisation of hierarchically scheduled multiprocessor embedded systems. *International Journal of Parallel Programming*, 36(1):37–67, 2008.

[Ram07] James W. Ramsey. Integrated modular avionics: Less is more, February 2007.

[Red04] Ola Redell. Analysis of tree-shaped transactions in distributed real-time systems. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pages 239–248, Washington, DC, USA, 2004. IEEE Computer Society.

[RGPH11] Juan M. Rivas, J. Javier Gutierrez, J. Carlos Palencia, and Michael González Harbour. Schedulability analysis and optimization of heterogeneous edf and fp distributed real-time systems. In *Proceedings of the 2011 23rd Euromicro Conference on Real-Time Systems*, ECRTS '11, pages 195–204, Washington, DC, USA, 2011. IEEE Computer Society.

[Rus99] John Rushby. Partitioning for avionics architectures: Requirements, mechanisms, and assurance. NASA Contractor Report CR-1999-209347, NASA Langley Research Center, June 1999.

[SBH+09] W. Steiner, G. Bauer, B. Hall, M. Paulitsch, and S. Varadarajan. TTEthernet Dataflow Concept. In *Proceedings of the Eighth IEEE International Symposium on Network Computing and Applications*, 2009.

[SKKS11] Till Steinbach, Hermand Dieumo Kenfack, Franz Korf, and Thomas C. Schmidt. An extension of the omnet++ inet framework for simulating real-time ethernet with high accuracy. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, SIMUTools '11, pages 375–382, ICST, Brussels, Belgium, Belgium, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[SRF09] Jean-Luc Scharbarg, Frédéric Ridouard, and Christian Fraboul. A probabilistic analysis of end-to-end delays on an avionics switched ethernet. *IEEE Trans. Industrial Informatics*, 5(1):38–49, 2009.

[Ste10a] Wilfried Steiner. An Evaluation of SMT-based Schedule Synthesis For Time-Triggered Multi-Hop Networks. In *Proceedings of the Real-Time Systems Symposium*, pages 375–384, 2010.

[Ste10b] Wilfried Steiner. Balanced-load planning of unsynchronized traffic for hard real-time networks. 2010.

[Ste10c] Wilfried Steiner. On the real-time performance of switches for rate-constrained multicast dataflow. Technical report, 2010.

[Ste11] W. Steiner. Synthesis of Static Communication Schedules for Mixed-Criticality Systems. In *Proceedings of the 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, 2011.

[TSMP12] Domiţian Tămaş-Selicean, Sorin Ovidiu Marinescu, and Paul Pop. Analysis and optimization of mixed-criticality applications on partitioned distributed architectures. In *Proceedings of the 7th International IET System Safety Conference*, 2012.

[TSP11a] Domiţian Tămaş-Selicean and Paul Pop. Design Optimization of Mixed-Criticality Real-Time Applications on Cost-Constrained Partitioned Architectures. In *Proceedings of the Real-Time Systems Symposium*, pages 24–33, 2011.

[TSP11b] Domiţian Tămaş-Selicean and Paul Pop. Optimization of time-partitions for mixed-criticality real-time distributed embedded systems. In *Proceedings of the 2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, ISORCW '11, pages 1–10, Washington, DC, USA, 2011. IEEE Computer Society.

[TSP12] Domiţian Tămaş-Selicean and Paul Pop. Synthesis of communication schedules for TTEthernet-based mixed-criticality systems. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, 2012.