Buffer Analysis and Message Scheduling for Real-Time Networks

vom
Fachbereich Elektrotechnik und Informationstechnik
der Technischen Universität Kaiserslautern
zur Verleihung des akademischen Grades eines

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von
Rodrigo Ferreira Coelho
geboren in Rio de Janeiro, Brasilien

D 386

Eingereicht am:	25.10.2017
Tag der mündlichen Prüfung:	21.11.2017
Dekan des Fachbereichs:	Prof. Dr.-Ing. Ralph Urbansky
Promotionskomission	
Vorsitzender:	Prof. Dr.-Ing. Wolfgang Kunz
Berichterstattende:	Prof. Dipl.-Ing. Dr. Gerhard Fohler
	Prof. Christian Fraboul
	Prof. Jean-Dominique Decotignie

Erklärung gem. § 6 Abs. 3 Promotionsordnung

Ich versichere, dass ich diese Dissertation selbst und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt und die aus den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Dissertation wurde weder als Ganzes noch in Teilen als Prüfungsarbeit für eine staatliche oder andere wissenschaftliche Prüfung eingereicht. Es wurde weder diese noch eine andere Abhandlung bei einem anderen Fachbereich oder einer anderen Universität als Dissertation eingereicht.

(Ort, Datum) (Rodrigo Ferreira Coelho)

Abstract

For many years, most distributed real-time systems employed data communication systems specially tailored to address the specific requirements of individual domains: for instance, Controlled Area Network (CAN) and Flexray in the automotive domain, ARINC 429 [FW10] and TTP [Kop95] in the aerospace domain. Some of these solutions were expensive, and eventually not well understood.

Mostly driven by the ever decreasing costs, the application of such distributed real-time system have drastically increased in the last years in different domains. Consequently, cross-domain communication systems are advantageous. Not only the number of distributed real-time systems have been increasing but also the number of nodes per system, have drastically increased, which in turn increases their network bandwidth requirements. Further, the system architectures have been changing, allowing for applications to spread computations among different computer nodes. For example, modern avionics systems moved from federated to integrated modular architecture, also increasing the network bandwidth requirements.

Ethernet (IEEE 802.3) [iee12] is a well established network standard. Further, it is fast, easy to install, and the interface ICs are cheap [Dec05]. However, Ethernet does not offer any temporal guarantee. Research groups from academia and industry have presented a number of protocols merging the benefits of Ethernet and the temporal guarantees required by distributed real-time systems. Two of these protocols are: Avionics Full-Duplex Switched Ethernet (AFDX) [AFD09] and Time-Triggered Ethernet (TTEthernet) [tim16]. In this dissertation, we propose solutions for two problems faced during the design of AFDX and TTEthernet networks: avoiding data loss due to buffer overflow in AFDX networks with multiple priority traffic, and scheduling of TTEthernet networks.

AFDX guarantees bandwidth separation and bounded transmission latency for each communication channel. Communication channels in AFDX networks are not synchronized, and therefore frames might compete for the same output port, requiring buffering to avoid data loss. To avoid buffer overflow and the resulting data loss, the network designer must reserve a safe, but not too pessimistic amount of memory of each buffer. The current AFDX standard allows for the classification of the network traffic with two priorities. Nevertheless, some commercial solutions provide multiple priorities, increasing the complexity of the buffer backlog analysis. The state-of-the-art AFDX buffer backlog analysis does not provide a method to compute deterministic upper bounds

for buffer backlog of AFDX networks with multiple priority traffic. Therefore, in this dissertation we propose a method to address this open problem. Our method is based on the analysis of the largest busy period encountered by frames stored in a buffer. We identify the ingress (and respective egress) order of frames in the largest busy period that leads to the largest buffer backlog, and then compute the respective buffer backlog upper bound. We present experiments to measure the computational costs of our method.

In TTEthernet, nodes are synchronized, allowing for message transmission at well defined points in time, computed off-line and stored in a conflict-free scheduling table. The computation of such scheduling tables is a NP-complete problem [Kor92], which should be solved in reasonable time for industrial size networks. We propose an approach to efficiently compute a schedule for the TT communication channels in TTEthernet networks, in which we model the scheduling problem as a search tree. As the scheduler traverses the search tree, it schedules the communication channels on a physical link. We presented two approaches to traverse the search tree while progressively creating the vertices of the search tree. A valid schedule is found once the scheduler reaches a valid leaf. If on the contrary, it reaches an invalid leaf, the scheduler backtracks searching for a path to a valid leaf. We present a set of experiments to demonstrate the impact of the input parameters on the time taken to compute a feasible schedule or to deem the set of virtual links infeasible.

To my parents, brother, wife and kids.

You cannot teach a man anything;
you can only help him discover it in himself.

Galileo Galilei

Preface

It is Autumn 2017, a beautiful time of the year. And also a good time to remember everyone that helped me on this long way to my PhD thesis.

First of all, I would like to thank Prof. Fohler, who certainly is a very good sample of his own description of the adjective Austrian: a mixture of German and Italian. Considering the nationalities clichés, this mix could go very wrong if we take, for instance, Italian punctuality and German spontaneity. On the other hand, the right mixture can be very positively inspiring. During the last years, I learned not only from our serious scientific discussions, but also from his positive manner of addressing hard problems. And also, that in some occasions, there is no better help than a beer with friends.

Further, I would like to thank Prof. Jean-Dominique Decotignie, Prof. Christian Fraboul and Prof. Wolfgang Kunz for being part of my PhD committee, and for providing valuable comments that helped to improve the contributions of this thesis.

During the years of my PhD studies, I met many colleagues at the university. Many of them became good friends. I am glad I had the opportunity to met you guys. This time would not have been as productive and fun without you, my former colleagues Alexander Neundorf, Cuong Ngo, Jens Theis, Mitra Nasri, Ramon Oliver, Raphael Guerra, and Stefan Schorr; and current colleagues Ali Syed, Ankit Agrawal, Florian Heilman, Gautam Gala, Kristin Krüger and Steven Dietrich.

I am also very thankful for three important people that make sure that the chair of Real-Time Systems runs seamlessly: Steffi, Markus, and Carmen. Despite not being officially in this chair, Carmen always helped me in important moments when Steffi was not here: thanks Carmen. Thanks Steffi for fixing the paperwork and giving me the right hints on the bureaucracy involved in my academic life: thanks to you, I could enjoy the benefits offered by the university and could still focus on my research. Thanks Markus, our technical support Chuck Norris. Both technically and personally, one of the most important persons in the group.

I had the great opportunity to supervise many undergraduate students, who collaborated to this thesis with good discussions, creation of many tools and some beers. Thanks Alan Green, Anoop Bhagyanath, Catalin Voinea, Jose Romero, Luiz Gonzaga, Manjula Kalloli, Pramod Murthy, Naga Rajesh and Vijay Desai.

Pursuing the PhD gave me the opportunity to meet great people also outside our research group in Kaiserslautern. Thanks Jean-Luc Scharbarg, Jirka Klaue, Sergio Penna and Stefan Schneele: our technical discussions greatly motivated me and contributed to

the results presented in this thesis.

And, above all, I am thankful to my family: indubitably, they have been my big strength all my life. Thanks *pai* and *mãe* for supporting me all my life. Not only showing me the importance of a good education, but also for making our home the most pleasant place to be. The distance between us did not stop them to take care of me in the hard times. I am also very thankful to my brother, for being a great motivator, my best friend and for supporting our parents during all these years while I have been abroad.

I am specially thankful to my wife. I can not find an appropriate adjective to define this great woman. Thank you very much Tatiana for supporting me all this time, for taking care of our home, for withstanding the last months in which I have spent very little time at home, and for raising our kids better than anyone else could ever do. I love you.

At last, and certainly not least, I am very thankful to my kids Rafael and Mathias. I would never have made it that far without the strength gained in the countless bad slept nights and clinic visits. These memories, on the other hand, quickly go away and are replaced by an immense happiness every time I see these boys laughing, playing and discovering the world around them. I love you kids.

Thank you all.

Publications

I have authored or co-authored the following publications:

Conference and refereed workshop papers

- Coelho, Rodrigo F. ; Fohler, Gerhard ; Scharbarg, Jean-Luc: Upper Bound Computation for Buffer Backlog on AFDX Networks with Multiple Priority Virtual Links. In: *32nd ACM Symposium on Applied Computing*. Marrakesh, Morocco : ACM Publisher, April 2017

- Coelho, Rodrigo F. ; Fohler, Gerhard ; Scharbarg, Jean-Luc: Dimensioning buffers for AFDX networks with multiple priorities virtual links. In: *Digital Avionics Systems Conference*. Prague : IEEE Computer Society, September 2015

- Coelho, Rodrigo F. ; Szczepanski, Mark ; Miari, Tarek: A web monitoring tool for AFDX networks. In: *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS) in conjunction with ECRTS 15*, 2015

- Coelho, Rodrigo F. ; Fohler, Gerhard ; Scharbarg, Jean-Luc: Worst-case backlog for AFDX network with n-priorities. In: *13th Workshop on Real-time Networks (RTN'14) in conjuction with 26th ECRTS*, 2014

- Garikiparthi, Naga R. ; Coelho, Rodrigo F. ; Fohler, Gerhard: Calculation of Worst Case Backlog for AFDX Buffers with Two Priority Levels using Trajectory Approach. In: *12th Workshop on Real-time Networks (RTN'13) in conjuction with 25th ECRTS*, 2013

- Coelho, Rodrigo F. ; Kotra, Anand ; Fohler, Gerhard: A Control Theory Approach to Video Stream Adaptation for Restricted Bandwidth Networks. In: *Proceedings of 1st Workshop on Adaptive Resource Management (WARM10)*. Stockholm, Sweden, April 2010

- Coelho, Rodrigo F. ; Fohler, Gerhard: A control theory based method for increased resource efficiency in real-time MPEG-2 video stream adaptation. In: *Proceedings of Work-in-Progress Session, 29th IEEE Real-Time Systems Symposium*, 2008

Contents

List of Figures

List of Tables

Chapter I

Introduction

For many years, most distributed real-time systems employed data communication systems specially tailored to address the specific requirements of individual domains: for instance, Controlled Area Network (CAN) and Flexray in the automotive domain, ARINC 429 [FW10] and TTP [Kop95] in the aerospace domain. Some of these solutions were expensive, and eventually not well understood. In recent distributed real-time systems the number of nodes per system, have drastically increased, which in turn increases the network bandwidth requirements. Further, the system architectures have been changing allowing for applications to spread their computations among different computer nodes. For example, modern avionics systems moved from federated to integrated modular architecture, also increasing the network bandwidth requirements.

Ethernet (IEEE 802.3) [iee12] is a well established network standard. Further, it is fast, easy to install, and the interface ICs are cheap [Dec05]. However, Ethernet does not offer any temporal guarantee. In order to close this gap, research groups from academia and industry have presented a number of protocols merging the benefits of Ethernet and the temporal guarantees required by distributed real-time systems. In the aerospace domain, two of these protocols are: Avionics Full-Duplex Switched Ethernet (AFDX) [AFD09] and Time-Triggered Ethernet (TTEthernet) [tim16].

AFDX guarantees bandwidth separation and bounded transmission latency for each communication channel. Communication channels in AFDX networks are not synchronized, and therefore frames might compete for the same output port, requiring buffering to avoid data loss. In order to avoid buffer overflow and the resulting data loss, computing an upper bound for the backlog in every buffer on the network is vital for the correct behavior of distributed real-time avionics systems. Although the current AFDX standard allows for the classification of the network traffic with only two priorities, some commercial solutions provide multiple priorities, increasing the complexity of the buffer backlog analysis.

In TTEthernet, nodes are synchronized, allowing for message transmission at well defined points in time, computed off-line and stored in a conflict-free scheduling table. The computation of such scheduling tables is a NP-complete problem [Kor92], which should be solved in reasonable time for industrial size networks.

In this dissertation, we address the two aforementioned problems faced during the design of AFDX and TTEthernet networks: avoiding data loss due to buffer overflow

in AFDX networks with multiple priority traffic, and computing the schedule of time-triggered communication channels in TTEthernet networks.

We address the first problem by presenting a method to compute an upper bound for the buffer backlog of each buffer on an AFDX network. This computation is based on the analysis of the largest busy period encountered by frames stored in a buffer. Our approach first identifies the ingress (and respective egress) order of frames that leads to the largest buffer backlog. Then, we compute the respective buffer backlog upper bound.

In order to efficiently compute a schedule for the TT communication channels in TTEthernet networks, we propose an approach in which we model the scheduling problem as a search tree. As the scheduler traverses the search tree, it schedules the communication channels on a physical link. Our scheduler is based on two approaches to traverse and, to progressively create the vertices of the search tree. A valid schedule is found once the scheduler reaches a valid leaf. If on the contrary, it reaches an invalid leaf, the scheduler backtracks searching for a path to a valid leaf.

We present in Section I.1, the basic concepts on distributed real-time systems used in this dissertation. Section I.2 describes the properties of real-time communication systems, and fundamental characteristics of two of these communication systems: AFDX and TTEthernet. Section I.3 presents an informal description of the problems addressed in this dissertation. Section I.4 describes the contributions proposed in this dissertation, and Section I.5 shows the organization of the next chapters.

I.1 Real-Time Systems

For *real-time* systems, not only is the correct value of the computation (logic correctness) important, but also the point in time when this value is available (temporal correctness). Logic and temporal correctness determine the quality of a real-time system. The classic example of an airbag depicts such a system: the airbag must inflate within a predefined time window after a car crash occurs. If it inflates too late, the passenger hits the dashboard before the airbag can provide any protection. If, on the contrary, it inflates too early, the airbag does not offer the optimal cushioning to the passenger.

One of the most important properties that a hard real-time system should offer is *predictability* [SR90]. With respect to predictability of real-time systems, Buttazzo describes in [But05]:

> "To guarantee a minimum level of performance, the system must be able to predict the consequences of any scheduling decision. If some task cannot be guaranteed within its time constraints, the system must notify this fact in advance, so that alternative actions can be planned in time to cope with the event."

The literature classifies real-time systems with respect to the impact on the environment in case of deadlines misses. Kopetz concisely describes this classification in [Kop11]:

> "If a result has utility even after the deadline has passed, the deadline is called *soft*, otherwise, it is *firm*. If severe consequences could result if a firm deadline is missed, the deadline is called *hard*. A real-time computer system that must meet at least one hard deadline is called a *hard real-time computer system*... If no hard deadline exists, then the system is called a *soft real-time computer system*"

Hard real-time systems must ensure that no deadline is missed. Therefore, such systems enforce the rules determining which *task* executes at which point in time: these rules define the schedule of a system. The creation of a schedule for a hard real-time system requires the knowledge about the characteristics of the executing tasks. These characteristics are described by a task model: an abstraction used to express the temporal requirements of tasks. Two common task models are the *periodic* and *sporadic* task models. In the first, a *task* (τ_i) is activated periodically, with period T_i, for the execution of a *job* ($j_{i,k}$). The exact time required by a job to execute is not known in advance. Therefore, task models provide the *worst case execution time* (WCET_i) as an upper bound for execution time of the jobs. Further, the latest acceptable completion time of a job of task τ_i is represented by the task deadline (d_i). The sporadic task model is similar to the periodic, except that the activations of jobs do not necessarily occur periodically. Instead of a period, the sporadic task model defines the minimum time between two consecutive job activations.

I.1.1 Distributed Real-Time Systems

In distributed real-time systems, the computed response depends on the interaction of interconnected computer nodes. Again, the correctness of a computation in such systems depends on the logical and temporal correctness. The time required for these nodes to communicate directly affects the response time of a task. Therefore, communication in distributed real-time systems is done via a *real-time communication system*, in this dissertation also referred to as *real-time network*. This type of communication system should add as little time as possible to the task response time. In other words, a real-time network should offer *low message transport latency*. Additionally, the transmission latency of a message should vary as little as possible, i.e., add *low jitter*. In a classic example of a control loop implemented as distributed real-time system, the control algorithm executes periodically, acquiring the state of the system under control (plant) by reading the value from a sensor, computing the control signal, and sending the computed value to an actuator. The control rules account for the physical effects on the plant considering a predefined latency caused by the network. In this example, if the transmission latency of the messages sent by the computer node to the actuator deviates from the expected latency, the plant will be in a state different from the one considered by the control algorithm, when the message arrives on the actuator. This unexpected latency variation could lead to system instability.

I.2 Real-Time Data Communication

According to [Kop11], data communication is classified in three fundamental categories: event-triggered (ET), rate-constrained (RC) and time-triggered (TT).

Event-Triggered Communication In ET communication, a message is sent every time a relevant event occurs, e.g., completion of a computation or change of an input signal. This type of communication allows every node to send a message at any point in time: in the worst case, every node tries to send a message at the same time. Three approaches handle this scenario: i) the network buffers these messages, ii) the network exerts back-pressure on the sending node, or iii) the network discards some messages. Consequently, ET communication may lead to large transmission latency jitter or even message losses.

Two examples of event-triggered networks are switched Ethernet (IEEE 802.3) and Controlled Area Network (CAN). Even though extensions to switched Ethernet providing real-time properties exist, switched Ethernet as defined in the IEEE standard 802.3 does not provide any temporal guarantee. In CAN, the protocol imposes backpressure flow control to the sender nodes by applying CSMA/CR (carrier sense multiple access/ collision resolution) in conjunction with message priority assignments: if more than one node try to send messages simultaneously, only the message with higher priority is transmitted. After this transmission is completed, the nodes try to send their messages again; as in the previous round, the message with higher priority among the competing messages is transmitted. This strategy imposes large jitter on messages with low priorities. Nevertheless, it is possible to provide an off-line analysis of the best and worst-case

transmission latency for every message in CAN.

Rate-Constrained Communication In RC communication, the network ensures a predefined share of the total bandwidth to each communication channel. Further, the network applies backpressure to the sending nodes to ensure that each communication channel does not utilize more network bandwidth than the predefined value. Avionics full-duplex switched Ethernet (AFDX) is an example of RC communication system.

Time-Triggered Communication In TT communication, the transmission time of each frame is decided off-line and stored in a conflict-free scheduling table. Frames are transmitted periodically in each communication channel. Therefore, two values define the transmission time of each frame of a communication channel: phase and period. Phase determines the dispatching time of the first frame of a communication channel w.r.t. the start of the schedule, and period specifies the timespan between the start of the transmission of two consecutive frames.

For a correct behavior of TT networks, all participating nodes must share the same notion of time, i.e., this type of communication relies on a clock synchronization protocol to maintain all nodes synchronized. Only then, can the participating nodes agree on the time of transmission and reception of frames. Time-Triggered Ethernet (TTEthernet) is an example of a time-triggered communication system.

I.2.1 Ethernet-Based Real-Time Communication

Intrinsically, Ethernet does not offer any temporal guarantees. Further, the correct delivery of frames can not be detected by protocol layers below the transport layer (it can only be detected at higher layers, or at the transport layer if TCP is used). Nevertheless, many solutions have been proposed in the last years to expand the IEEE standard 802.3 in order to provide temporal guarantees. Decotignie describes in [Dec05] how the intrinsic properties of Ethernet prohibit the use of standard Ethernet in real-time systems. He further presents different strategies to overcome these properties and refers to various existing solutions in the industrial domain.

Recently, other solutions have been proposed to modify the standard Ethernet to provide temporal guarantees. Particularly, the Time-Sensitive Network (TSN) IEEE task group, previously called Audio Video Bridging (AVB) task group, has been presenting solutions to allow for guaranteed packet transport with bounded low latency, low jitter, and low packet loss [Gro17].

In the last years, two Ethernet-based real-time communication systems have gained large attention in the industry: Avionics Full-Duplex Switched Ethernet (AFDX) and Time-Triggered Ethernet (TTEthernet).

AFDX AFDX is a rate-constrained network based on switched Ethernet, used in most recent aircraft. A *virtual link* defines a logical communication channel between one source and to one or more destinations. The properties of the frames of a virtual link are defined off-line, namely: i) a static route, ii) a priority level, iii) a maximum frame

size (L_{max}), iv) a minimum time between two consecutive frames (BAG), and v) a maximum jitter. AFDX applies traffic shaping and policing to avoid the propagation of unexpected traffic.

Due to unsynchronized message transmission, contention may occur on the output ports of switches. Nevertheless, since the traffic route and the dispatching rules are static and known before run-time, off-line analysis allow for the computation of upper bounds of the frame transmission latencies and buffer backlog.

TTEthernet TTEthernet is a real-time network based on switched Ethernet which allows for best-effort, rate-constrained and time-triggered traffic to coexist in the same communication system. TTEthernet provides a fault-tolerant transparent clock synchronization protocol, required to synchronize the nodes participating in the time-triggered message communication. Similarly to AFDX, TTEthernet applies traffic policing to avoid the propagation of unauthorized traffic.

I.3 Description of the Addressed Problems

In this dissertation, we propose solutions for two problems faced during the design of AFDX and TTEthernet networks. The following sections describe those problems.

I.3.1 Avoiding Data Loss due to Buffer Overflow in AFDX Networks with Multiple Priority Traffic

In AFDX networks, the transmission of frames is not synchronized. Thus, the points in time when the frames ingress a switch are not known in advance. As a result, frames of different virtual links might compete for a switch output port, leading to data contention. AFDX addresses this data contention by means of buffering.

The ARINC 664 Part 7 standard (AFDX) [AFD09] states that "each output port of the switch should be able to buffer at least 512 frames (balanced between high and low priority)." Despite being a very specific recommendation, the actual largest buffer backlog depends on the traffic on the network. Further, the AFDX standard does not define how the actual memory reservation should be distributed among buffers of different priorities. Therefore, deciding on the amount of memory reserved for the buffer of each priority level is left as a design decision.

In order for the designer to provide a safe, but not too pessimistic value for the memory allocated to each buffer, he/she must compute an upper bound for the backlog of each buffer in the network. Note that, due to the safety critical properties of avionics systems, buffer overflow and the resulting data loss must be avoided at all cost. Nonetheless, large overprovision of memory to buffers should be avoided.

The current ARINC 664 Part 7 standard defines two priority levels that can be assigned to the network traffic. Nevertheless, some AFDX commercial products allow for the assignment of multiple priority levels. For instance, the AFDX switch described in [TTT], provides eight priority levels. In this dissertation, we also relax the two priority

levels limitation described in the current ARINC 664 Part 7 standard, and assume that the network traffic may be configured with any priority out of a set with multiple priority levels. Consequently, the network designer must configure the memory reservation for not only two, but for multiple buffers on each output port of every switch on the network.

Current methods to compute the exact largest buffer backlog rely on checking all possible combinations of ingress and egress times of frames. Therefore, computing the exact largest buffer backlog value of each buffer in an industrial size AFDX network is intractable. Considering the unpredicted dispatch time of AFDX frames, the computational requirements of those methods explode as the network traffic increases.

The aforementioned issues depict the importance and the challenges of computing an upper bound for the buffer backlog on AFDX networks with multiple priority traffic.

I.3.2 Scheduling of TTEthernet Networks

In time-triggered networks, the transmission time of each frame is known to the sender and receiver(s) before run-time. A scheduling table stores the transmission and reception time window for each frame. This table is computed off-line and ensures that transmission windows (on a physical link) do not overlap.

The start of the transmission window of each frame is defined by two parameters: *period* and *phase*, where *period* represents the constant amount of time between the transmission of any two consecutive frames of the same message, and *phase* represents the amount of time between the start of the schedule until the point in time when the first frame of a message is scheduled.

The computation of a scheduling table for TT messages is a NP-complete problem in the strong sense. Hence, the synthesis of a scheduling table for an industrial size TT network in reasonable time is a challenging problem.

I.4 Contributions

I.4.1 Computation of Buffer Backlog Upper Bounds for AFDX Networks with Multiple Priorities

The maximum backlog of a buffer on an output port of an AFDX switch depends on the frames that compete for that port (these frames are called *competing frames*). It depends not only on the number of competing frames but also on their priority, on their size, and on the order in which they ingress and egress the buffer.

In order to compute the backlog upper bound for a buffer on an AFDX output port, we initially identify the competing frames in the largest busy period of each virtual link that egress from that output port. Our approach then computes the *worst case scenario* for the busy period of each virtual link. We define *worst case scenario* as the ingress order (and resulting egress order) of competing frames that leads to the largest backlog of the buffer under analysis encountered by a frame of that virtual link on a given output port. Then, we compute the largest backlog encountered by a frame of that virtual link

on its worst case scenario. In the final step, we analyze the largest backlog encountered by a frame of each virtual link with the same priority as the buffer under analysis (and egressing through the same output port), and compute an upper bound for the backlog of that buffer.

The method presented in this dissertation extends the state-of-the-art by proposing a deterministic (not stochastic) computation of an upper bound for the buffer backlog in AFDX networks with multiple priority traffic.

I.4.2 A Strictly Periodic Scheduler for Time-Triggered Ethernet

In this dissertation we propose an off-line scheduler for time-triggered networks, in particular for TTEthernet, called Search Tree based Scheduler for Time-Triggered Networks (STSTTN). We formulate the scheduling problem as a set of search trees, each of them representing one physical link. As STSTTN traverses a search tree, it schedules one virtual link on each level by selecting a phase, represented by an edge in the tree. Once STSTTN reaches a leaf on the deepest level of the tree, a valid schedule is found. If STSTTN reaches an invalid leaf, it backtracks until a valid edge is found or the set of virtual links is deemed unschedulable.

We present two approaches to traverse, and progressively create the vertices of the search tree: *look back* and *look ahead*. The first approach checks for the feasibility of a selected phase based on the properties of all virtual links which have already been scheduled at that point of the search. Look ahead additionally accounts for the properties of virtual links which have not been scheduled yet. We further present an optional pruning tree heuristic to reduce the search space.

While traversing the search tree, STSTTN analyzes the properties of each virtual link allowing for an efficient selection of phases.

I.5 Dissertation Outline

The rest of this dissertation is organized as follows:

Chapter II – In this chapter, entitled *AFDX Background and State-of-the-Art in Buffer Backlog Analysis*, we describe the fundamental AFDX properties used throughout this dissertation and the related work on the computation of upper bounds for the worst case traversal time and buffer backlog.

Chapter III – In this chapter, entitled *Buffer Backlog Upper Bound for AFDX Networks with Multiple Priority Traffic*, we present a method to compute an upper bound for the backlog on each output buffer of AFDX networks with multiple priority traffic. We describe the properties of the worst case scenario for the buffer backlog, and present the formulas for the computation of an upper bound for the backlog of each buffer on an AFDX network. Additionally, we discuss how the values computed by our method can be applied to AFDX switches that do not conform with our assumptions.

Chapter IV – In this chapter, entitled *TTEthernet Background and State-of-the-Art in Time-Triggered Schedulers* we describe the properties of Time-Triggered Ethernet required by the scheduler proposed in Chapter V and present the state-of-the-art methods to compute schedules for time-triggered systems.

Chapter V – In this chapter, entitled *Off-Line Scheduler for Time-Triggered Networks*, we propose an off-line scheduler for Time-Triggered Ethernet networks, called Search Tree based Scheduler for Time Triggered Networks (STSTTN).

Chapter VI – In this chapter, we summarize the contributions of this dissertation and bring the concluding remarks.

Appendix A – In this appendix, we present the details of the proof of Property iii) of the Theorem III.1.

Appendix B – In this appendix, we show how to apply the trajectory approach to compute the competing frames used in the computation of the buffer backlog upper bound presented in Chapter III.

Chapter II

AFDX Background and State-of-the-Art in Buffer Backlog Analysis

In this chapter, we present the fundamental AFDX properties used throughout this dissertation and the related work on the computation of upper bounds for the worst case traversal time and buffer backlog. We start Section II.1 by presenting the origins of AFDX and how this network relates to previously deployed avionics networks. We present the types of devices connected to the network, explain the concept and the properties of virtual links, and show how network cross traffic leads to contention on the network.

Due to their safety critical nature, distributed avionic systems (in this dissertation, we use the terms *avionics* and *avionic systems* interchangeably) demand real-time behavior. We show in Section II.2, that despite the cross traffic, AFDX networks can guarantee predictable temporal properties, namely bounded worst case traversal time and no message loss, for the frames traversing the network. In order to compute upper bounds to the worst case traversal time and buffer backlogs, a number of algorithms have been proposed. We present in Section II.3 state-of-the-art methods to compute these upper bounds and discuss their advantages and limitations. Section II.4 concludes this chapter with a summary.

II.1 Introduction

The need for data communication in avionic systems exists since many decades. When electronic devices were introduced in commercial aircraft, the task of data communication was sending information from a variety of sensors (radar, engines etc) to cockpit displays. ARINC 429 [FW10] is one of the first standards (introduced in 1978) specifically designed for avionic systems and it is still vastly used in current commercial fleets. ARINC 429 uses a unidirectional data connection with transmission rates up to 100kbps. Data communication in ARINC 429 relies on a direct cable connection between source and destination(s) of each communication channel, i.e., each communication channel forms an individual connection with its own cables. For a communication channel with n destinations, ARINC 429 requires n point-to-point connections using n cables. Consequences of the excessive cabling required to establish the point-to-point communication channels are: a large impact on the aircraft weight and a high complexity involved in designing, deploying and maintaining such a system. Figure II.1 presents the physical interconnection of a simple avionic system using ARINC 429.

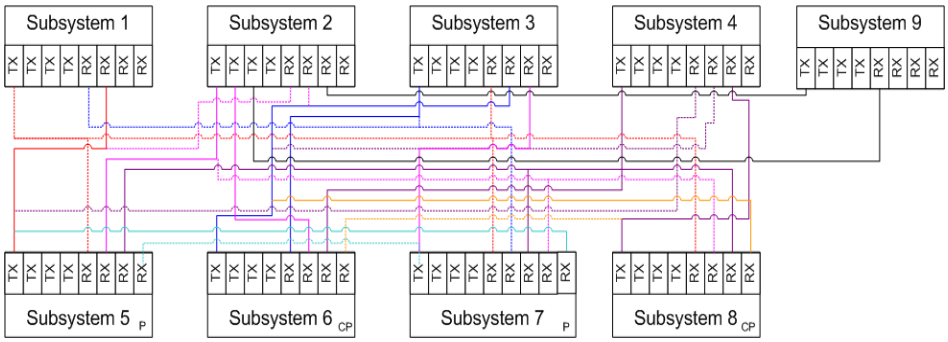


Figure II.1: *Physical interconnection of a simple avionic system using ARINC 429. Adapted from [SV08]*

ARINC 429 has been used in avionic systems based on federated architectures. In such architectures, each application is allocated to a dedicated hardware (Line Replacement Unit - LRU). This architecture leads to a number of issues. For example, in order to accommodate possible future functionalities added during the lifetime of an aircraft, each LRU is designed with large resource overprovision. Considering the overprovision of each LRU, the total resource (and cost) reserved in a federated architecture is large.

Most recent aircraft implement a new architecture paradigm retiring the federated architecture in favor of the Integrated Modular Avionics (IMA) systems. IMA allows for flexibility when allocating applications to resources. Thus, resources can be shared among multiple applications. On IMA systems, LRUs have been replaced with "avionics boxes" to host general purpose controllers, called CPIOM (Core Processing & IO Module) [But10]. In this type of architecture, reserved resources in CPIOMs can be used by multiple applications. Figure II.2 depicts the architecture difference for a simple example of an avionic system implemented in a federated and an IMA architecture.

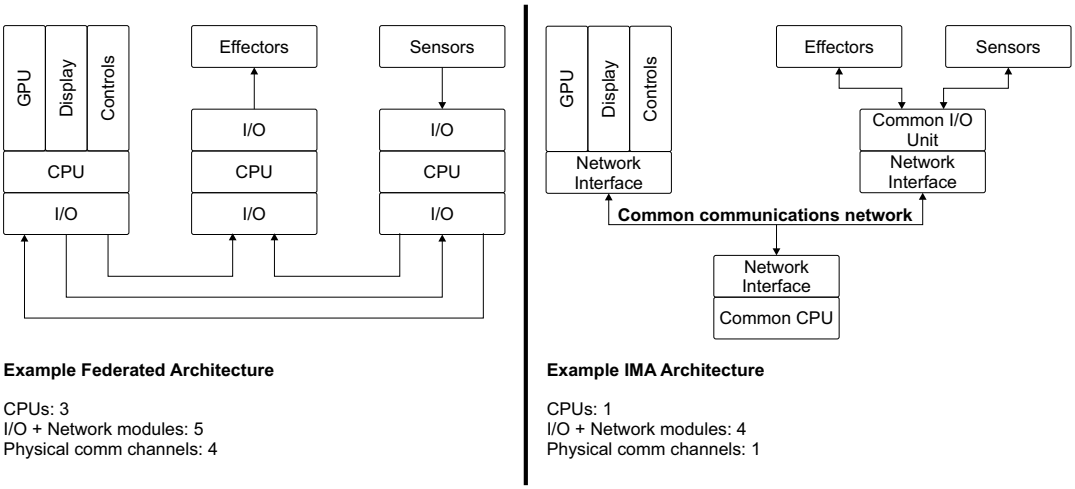


Figure II.2: *Comparison of federated and IMA architecture. Adapted from [WW07]*

Sharing resources among distinct (safety critical) applications requires complex timing analysis and methods to enforce the required temporal and spatial resource isolation. We refer the interested reader to the ARINC 653 standard "Avionics Application Software Standard Interface" for further details on how to ensure the temporal guarantees required by avionic systems.

ARINC 429 drawbacks and the communication requirements of IMA architectures made ARINC 429 obsolete. Therefore, EADS Airbus division developed the standard ARINC 664 Part 7 [AFD09] as *Avionics Full DupleX switched ethernet (AFDX)*. ARINC 664 Part 7 defines AFDX networks in different aspects. In this chapter, we focus on the parts of the standard relevant to this dissertation.

In essence, AFDX is an Ethernet based network tailored to account for the avionic systems requirements. On top of the advantages achieved with switched Ethernet, e.g., simpler cabling (compared to ARINC 429), high bandwidth, and no frame collision, AFDX provides a set of properties required by the safety critical applications running on avionic systems: redundancy, bandwidth isolation and temporal guarantees.

II.1.1 Devices

Two types of devices are used in AFDX networks: end-system (ESs) and switches. Figure II.3 shows, on the left-hand side, a sketch of the publicly available Airbus A380 AFDX network (Section II.1.2 explains the contents of the right-hand side). Each element in the actual network has a redundant counterpart which is not shown in this figure. Figure II.3 presents end-systems as circles and switches as rectangles.

According to the ARINC 664 Part 7 standard, the main function of the end-system is to provide services (interface) which guarantee a secure and reliable data exchange

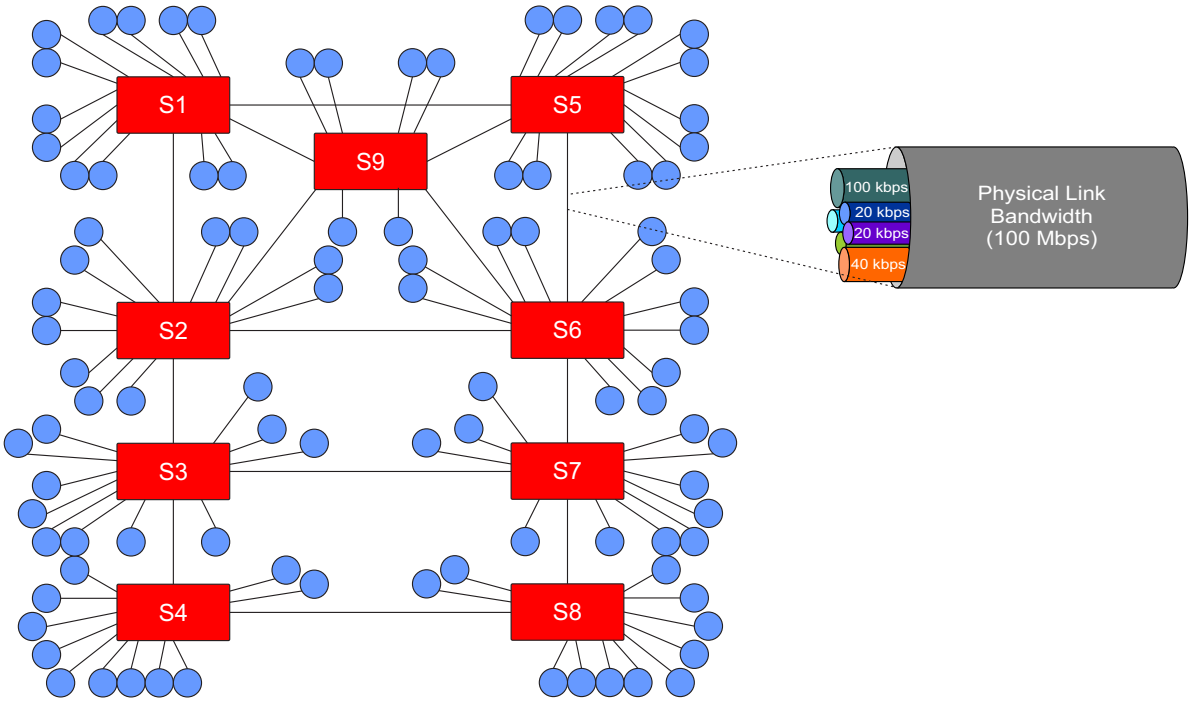


Figure II.3: *Publicly available AFDX topology of the Airbus A380. On the right-hand side, the relationship between physical link and bandwidth of virtual links.*

to the software running on AFDX nodes. Any two AFDX nodes are connected via a AFDX switch. These switches apply the store and forward paradigm to relay frames from input to output ports.

II.1.2 Virtual Link

The concept of virtual link (VL) has its origin in the communication channels used by ARINC 429. A virtual link defines a logical communication link between one source application located in a node to one or more destination applications in one or more nodes. To each virtual link, a share of the total physical link bandwidth is guaranteed. In a physical link connecting two nodes, one can imagine that a set of independent communication channels (virtual links) exist, each of which with its own share of the total bandwidth. Figure II.3 shows, on the right-hand side, a representation of how virtual links bandwidth reservations relate to the maximum bandwidth of a physical link (this example highlights the physical link connecting switches S5 and S6).

AFDX specifies two parameters to enforce the bandwidth isolation guaranteed to each virtual link: a bandwidth allocation gap (BAG) and the maximum size of an Ethernet frame (L_{max}). BAG represents the minimum time interval between the first bit of two consecutive frames from the same virtual link, assuming no jitter imposed by the end-system scheduler. AFDX defines eight possible values for BAG: 2^n ms, where $n \in \{0, 1, 2, 3, 4, 5, 6, 7\}$. L_{max} is the size, in bytes, of the largest Ethernet frame that can be transmitted over this virtual link. The bandwidth allocated to a virtual link is

given by the ratio $\frac{L_{max}}{BAG}$.

End-system schedulers are responsible for scheduling the transmission of frames of all egress virtual links: they not only throttle the dispatch of frames of a virtual link to enforce the minimum frame separation imposed by BAG (traffic regulator), but also multiplex the dispatch of frames of all egress virtual links (scheduler multiplexer). Figure II.4 depicts the role of end-system traffic regulators and multiplexers. Notice that AFDX switches perform filtering and traffic policing to avoid unexpected traffic to cross the network. A frame is delayed by the scheduler multiplexer every time two or more frames enter the multiplexer at the same time. This delay is called jitter. In Figure II.4, a frame of VL2 is delayed by a frame of VL1, and a frame of VL3 is delayed by a frame of VL1 and VL2 (the figure highlights the jitter suffered by a frame of VL3). In order to ensure predictable bounds for frame traversal time, the maximum allowed jitter for each virtual link is defined during the design phase of the network. During the network design phase, a static route taken by (frames of) each virtual link is defined.

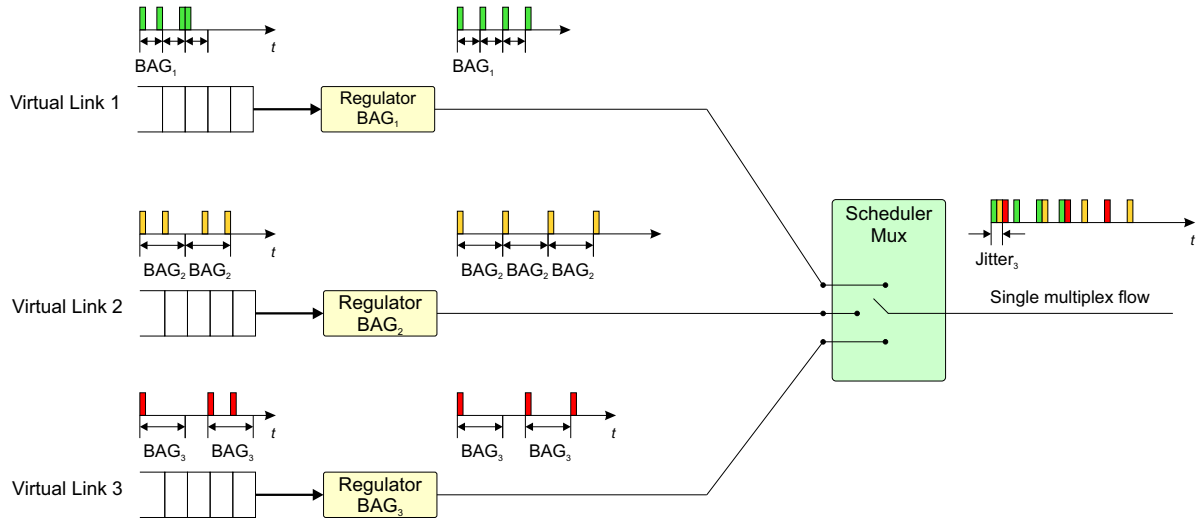


Figure II.4: *AFDX end-system regulator and scheduler multiplexer*

According to ARINC 664 Part 7, each virtual link is classified with one priority: *high* or *low*. In case of contention for an output port (see Section II.1.3), priorities associated to a frame define which frame egresses the respective port first, i.e., frames of *low* priority can only egress an output port if no frame of *high* priority is ready to egress this port. The current ARINC 664 Part 7 standard defines two priority levels for virtual links. Some commercial AFDX products, however, allow for the assignment of multiple priority levels to virtual links. The AFDX switch presented in [TTT], for instance, permits the classification of virtual links with 8 priorities. Remember that AFDX defines 8 values for BAGs. Thus, assigning virtual links priorities according to their BAGs such that the lower the BAG the higher the priority, allows for the generation of a non-preemptive rate monotonic frame schedule.

II.1.3 Frame Relay, Contention and Buffering

AFDX end-systems do not share any notion of global time and are, consequently, not synchronized. Therefore, frames of virtual links ingressing a switch from different input ports and egressing from the same output port might compete for this output port leading to data contention. ARINC 664 Part 7 addresses the data contention issue by means of buffering, i.e., each switch output port stores in an output buffer all the frames that cannot be directly forwarded to the output port. The impact of buffering on the performance of an AFDX network is twofold: first, buffering delays the transmission time of frames traversing the network; second, buffer overflow leads to frame losses, which should be avoided at all cost in safety critical avionic systems.

II.2 Predictable Properties

To ensure real-time behavior, required by safety critical applications on avionic systems, AFDX network designers must ensure that: i) the delay imposed on frames traversing the network is lower than the accepted delay for that virtual link, and ii) the output buffers do not overflow leading to frame losses. These two properties are not intrinsically guaranteed by AFDX networks. Rather, AFDX networks provide the deterministic mechanisms that allow for achieving these properties:

i) frames arrival in the same order they are transmitted (for each virtual link)

ii) bounded technological latency (also called switching latency – sl).

iii) bounded transmission jitter on end-systems (Section II.1.2)

iv) bounded transmission jitter on switches (configuration parameter [AFD09] Section 4.1.1.1)

v) bounded maximum delay caused by a switch (*"maximum elapsed time between: ingress of the last bit of a frame on the input port of a switch and egress of this last bit of the frame from the given output port of the switch"* [AFD09])

Nevertheless, the correct off-line temporal analysis allows for the network designer to ensure that both the frame delay and buffer backlog do not exceed the respective expected values. We refer to the time required for a frame to traverse the network, from its source end-system to its destination end-system as *traversal time*, *transmission latency*, and *end-to-end delay*, interchangeably.

II.2.1 Frame Delay

We divide the analysis on delay encountered by a frame traversing an AFDX network in two parts: first, the latency imposed to a frame during transmission and reception disregard any other frame on the network; second, the latency imposed by transmission/ reception of other frames. For the first part, values depend on the hardware used

in the network and the upper bounds are defined in ARINC 664 Part 7. The latter requires thorough analysis of all virtual links on the network. Investigating all possible combinations of contention on each output port is intractable for AFDX networks used in large commercial airplanes. Also, a naive approach assuming a critical instance as the time when the first frame of all virtual links sharing the same output port arrive at the same time, leads to extremely pessimistic values since this scenario is not achievable in all cases. We present in Section II.3, state-of-the-art methods used to compute upper bounds for the delay encountered by a frame in an AFDX network.

II.2.2 No Message Loss

AFDX offers two mechanisms to avoid frame losses on the network: redundancy and buffering. An AFDX network is composed of two redundant networks: the so-called A-network and B-network. In case of frame losses, for instance in the A-network, a copy of this frame arrives at the destination end-system through the B-network. Presentation of the redundancy mechanisms used in AFDX networks is out of the scope of this dissertation. For details on redundancy management on AFDX networks, we refer the interested reader to the ARINC 664 Part 7 standard [AFD09].

As presented in Section II.1.3, ARINC 664 Part 7 addresses the contention on the output port of switches by buffering. The standard states that "*each output port should be able to buffer at least 512 frames (balanced between high and low priority)*". Despite being a very specific recommendation, the number of 512 frames does not imply a safe buffer length for all possible traffic configurations of an AFDX network. Further, the standard does not define how the actual memory allocation for each buffer should be done. Therefore, dimensioning the output port buffer size for each priority level is left as a design decision. These values must be passed as parameters during the configuration of the network (section 4.7.3.2 of ARINC 664 Part 7). We present in Section II.3 the state-of-the-art methods used to compute upper bounds for the buffer backlog encountered by a frame in an AFDX network.

II.3 Off-Line Analysis to Ensure Predictable Properties

At run-time, AFDX provides the mechanisms (frame schedule, filtering) to enforce the predictable properties presented in Section II.2. However, it is up to the network designer to ensure that once the AFDX policies are applied at run-time, all temporal requirements are met. Therefore, a collection of off-line methods (tools) exists to allow the network designer to configure the AFDX network such that the network communication does not lead to violation of the applications temporal requirements. As mentioned in Section II.2, simulating all possible frame arrival combinations to achieve exact numbers for the largest delay encountered by a frame traversing an AFDX network or for the largest backlog of an output buffer, is intractable for larger avionics networks in commercial aircraft. The main reason for this highly time consuming analysis is the unsynchronized and sporadic nature of AFDX flows, i.e., the points in time when frames are transmitted on the network are not known before run-time, rather the minimum

interval between these frames and their maximum sizes are known. Therefore, most of the methods presented in this section compute upper bounds, not maximum achievable values, for frames of a virtual link traversing the network, and for the buffer backlog.

The state-of-the-art methods presented in this section can be divided into two categories: Methods of the first category analyzes the traffic as flows, assuming that both the traffic generated by virtual links and the network services provided by switches are bounded by so called *envelopes*. Network calculus (Section II.3.1) fits into this category. Methods of the second category analyze the network at a finer granularity: instead of considering a virtual link as a flow, these methods investigate the behavior of a selected frame of the virtual link under analysis. Later in this chapter we introduce five such methods: trajectory approach (Section II.3.2), forward end-to-end delay approach (Section II.3.3), simulation (Section II.3.4), model checking (Section II.3.5), and holistic approach (Section II.3.6).

II.3.1 Network Calculus

Network calculus (NC) provides a set of formal tools for modeling the network communication, including AFDX networks, and computing upper bounds for frame delays and buffer backlogs. Network calculus has been used for certification purposes, including the certification of the Airbus A380 [FFG06]. This method is based on two types of curves: arrival and service curves. The first is used to bound the traffic and the latter to bound the service offered by AFDX nodes. We present in this section a very short introduction to network calculus and concentrate on graphic explanations of this method. For details on the algebra behind network calculus, we refer the interested reader to [Cru91b], [Cru91a] and [LBT01].

II.3.1.1 Arrival Curves

Let us consider that the amount of data generated by a virtual link until a point in time t is represented by a cumulative function $x(t)$. An arrival curve α represents an upper bound on the data generated by this virtual link during a time span of length $t-s$, such that:

$$x(t) - x(s) \leq \alpha(t-s)$$

Figure II.5 depicts, on the left-hand side, an example of a cumulative function $x(t)$ constrained by an arrival curve α [LBT01]. The arrival function is presented separately on the right-hand side.

II.3.1.2 Service Curves

Service curves represent a lower bound on the guaranteed service offered by network nodes. Let us consider a flow represented by the cumulative function x traversing a

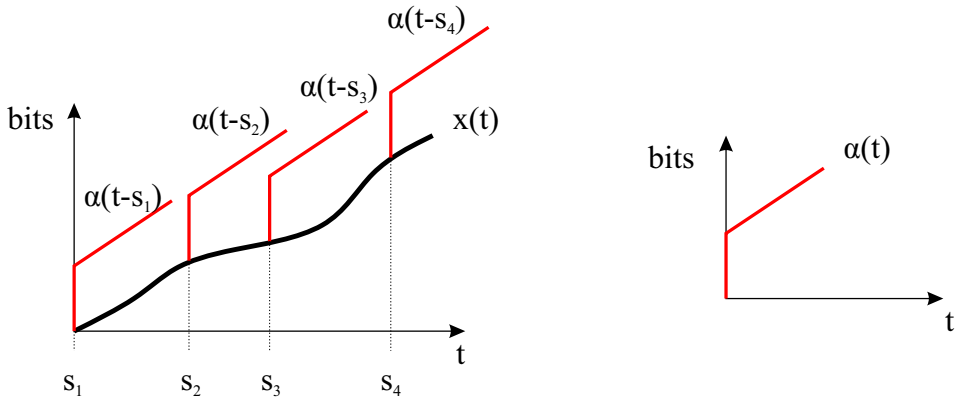


Figure II.5: *Example of an arrival curve. Adapted from [LBT01]*

system $\mathcal{S}$ where y represents the output flow leaving $\mathcal{S}$. We say that $\mathcal{S}$ offers to the flow x a service curve β, if and only if:

$$y(t) - x(t) \geq \beta(t - s)$$

where β is a wide sense increasing function, with $\beta(0) = 0$, for all $t \geq 0$ [LBT01]. Figure II.6 depicts an example of a cumulative function $y(t)$ and a lower bound represented by a service curve β [LBT01].

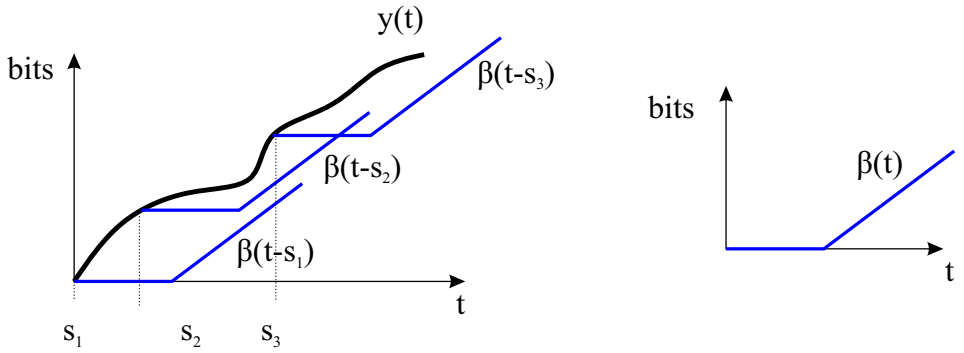


Figure II.6: *Example of a service curve. Adapted from [LBT01]*

II.3.1.3 Delay and Backlog

Figure II.7 depicts two curves commonly used to bound respectively, the traffic generated by a flow and the service offered by a network node: the *leaky bucket* arrival curve ($\gamma_{r,b}$) and the *rate latency* service curve ($\beta_{R,T}$).

Two parameters define a leaky bucket curve: r represents the bandwidth, and b represents the largest data burst. For the service curve $\beta_{R,T}$, the parameter R represents

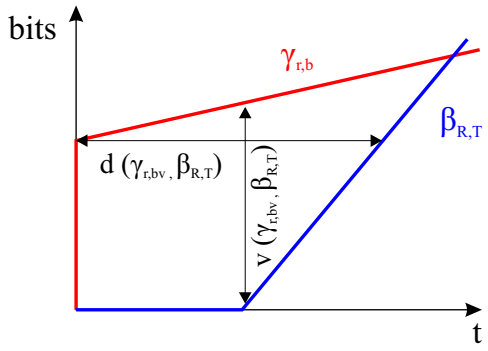


Figure II.7: *Delay and backlog on a node. Adapted from [LBT01]*

the minimum sustained bandwidth offered by the node, and T represents the largest delay imposed on a flow traversing the node. We can extract two important pieces of information from Figure II.7: i) an upper bound for the largest delay imposed on flow x, and ii) an upper bound for the largest backlog on the system modeled by β. These values are computed, respectively, by calculating the largest horizontal and vertical distances between the arrival and service curves. These differences are represented by d and v in Figure II.7. Considering the convexity of the region between α and β in the curves depicted in Figure II.7, the values d and v are reached at angular points of either α or β [LBT01].

Notice that the computed values represent upper bounds, not the largest achievable values or the worst case values. The tightness of the upper bounds computed with network calculus strongly depends on the tightness of arrival and service curves, i.e., how close these curves represent the modeled traffic and service, respectively. On one hand, simple curves have well-known properties [BMN11], which simplify the network calculus computations. On the other hand, for some networks, simple curves lead to pessimistic bounds.

A large number of published papers integrate the properties of AFDX traffic and the frame scheduling (dispatching) policies into arrival and service curves. A common issue in the analysis using network calculus is that, originally, this approach does not take into account the so-called serialization effect. Serialization occurs when analyzing the impact of multiple flows arriving at a network node from the same input port. In this case, two frames of these flows can never arrive at the node at the same time. Consequently, the largest burst represented by b in the leaky bucket arrival curve, is not the sum of the frames of these two flows. This effect is neglected by the simple analysis of arrival and service curves. [BSF10] and [ZLX+13] present two methods to account for the serialization effect. The comparison presented in [BSF10] shows that, even for a small example network, accounting for the serialization effect decreases the pessimism by more than 10% when compared to the original network calculus.

The methods presented so far use *deterministic network calculus* (DNC) to compute deterministic upper bounds for delay and buffer backlog. Another approach called *stochastic network calculus* (SNC) [SRF02], [RSF08] computes upper bounds for end-to-end delays and buffer backlogs that occur in AFDX networks with a certain probability.

Results of an example presented in [RSF08] show that, in comparison to the deterministic approach, stochastic network calculus can be much less pessimistic. Another interesting result from this work is that the larger the network load, the larger is the difference between the end-to-end delay upper bounds computed by deterministic and stochastic network calculus. For instance, considering a probability of 1×10^{-4} that the end-to-end delay exceeds the SNC upper bound, SNC computes an upper bound equal to $106\mu s$, while DNC computes $2086\mu s$.

In summary, network calculus allows for the computation of upper bounds for worst case traversal time and buffer backlog values on AFDX networks. This approach has been used for certification purposes. Nevertheless, network calculus computations are pessimistic. On top of the pessimism due to the modeling of traffic and service curves as *envelopes*, i.e., not exact curves, the holistic nature of analysis using network calculus (intrinsically) adds pessimism to the computed upper bounds: the resulting upper bound is the sum of the upper bound on each node. In most networks, the conditions leading to the upper bounds in each node are unachievable.

II.3.2 Trajectory Approach

Martin and Minet introduce the trajectory approach (TA) in a technical report [MM04] and in two conference: papers [MM06a] and [MM06b]. These papers present the theory behind the trajectory approach, however they do not address the analysis of AFDX networks specifically. Bauer et al. present the applicability of TA in the analysis of AFDX networks in [BSF09]. The main contribution of the trajectory approach is to provide an upper bound for the end-to-end delay encountered by any frame of a flow traversing a data network. Unlike holistic approaches, the trajectory approach accounts for the worst case scenario experienced by a frame on its trajectory and not on each traversed node.

II.3.2.1 Assumptions

The underlying assumption of the trajectory approach is that no collision (ensured by AFDX) or frame losses occur on the network. We describe in the next sections, additional properties assumed in the trajectory approach. Figure II.8 depicts an example of the network model used by the trajectory approach.

II.3.2.1.1 Network In the trajectory approach, each node represents an output port (of a switch or an end-system), including the respective output physical link. Nodes are the entities responsible to dispatch frames according to the configured frame scheduling policy. Most of the published trajectory approach work assumes a Fixed Priority First In First Out (FP/FIFO) policy.

Network nodes are interconnected by "links", which represent the switch fabric. For the end-to-end delay analysis, the links account for the time taken by switches to forward the packets internally (from a switch input to an output port). The nomenclature used by TA, depicted in Figure II.8, is rather non-intuitive. That terminology has its origins

in the analysis of distributed systems where nodes are responsible for the computation and the connections between nodes exhibit a static behavior.

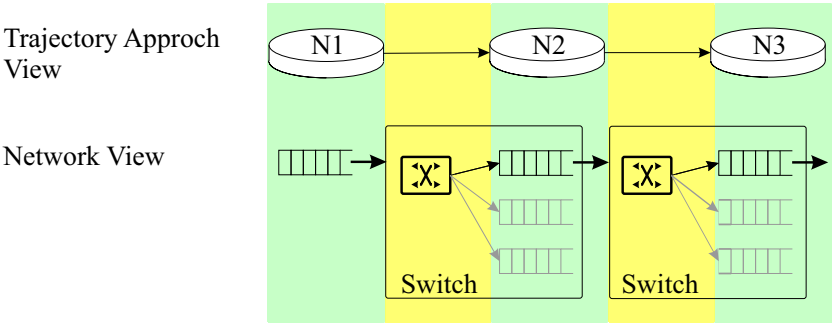


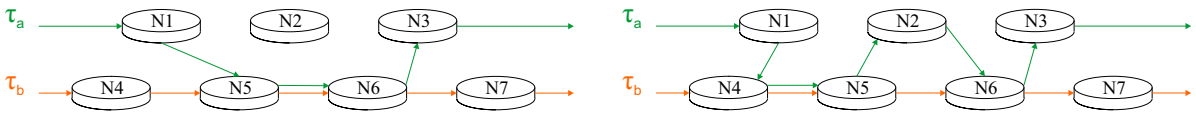
Figure II.8: *Nodes and links according to the model used by the trajectory approach*

II.3.2.1.2 Flows The trajectory approach assumes that flows generate frames periodically and might experience a release jitter. In the context of AFDX networks, these parameters represent the BAG and maximum jitter on the source end-system, respectively. For the trajectory approach, the "computation time" in a node represents the transmission time of a frame traversing that node, i.e., $\frac{L_{max}}{BAG}$.

II.3.2.1.3 Path The path of each flow, i.e., nodes traversed by a frame from its source to its destination, is static and defined off-line. The trajectory approach imposes a restriction on the valid flow paths: it assumes that if the path of a flow τ_i "meets" the path of a flow τ_j and "leaves" this common sub-path, the path of these two flows do not "meet" again. Figures II.9a and II.9b depict a valid and an invalid set of paths for two flows, τ_a and τ_b. Notice that, according to the mentioned TA restriction, Figure II.9b presents an invalid set of paths, since τ_a meets τ_b on node N4 and N5, then leaves the path of τ_b and meets τ_b again on node N6.

II.3.2.2 Computations

The trajectory approach computes an upper bound for the end-to-end traversal time encountered by any frame, e.g., f^m, of a flow τ_i by analyzing the busy period of this frame along its trajectory from source to destination node. On each node traversed by



(a) *TA valid path* **(b)** *TA invalid path*

Figure II.9: *Valid and invalid paths for τ_a and τ_b according to the trajectory approach assumptions*

f^m, the trajectory approach accounts for the delay accrued by each flow that "meets" the path of τ_i for the first time. TA defines the points in time M_i^h and S_{max}^h to represent the earliest start time of the busy period of f^m and the latest start time of arrival of f^m on node h, respectively. According to [BSF09], for a node dispatching frames according to the FIFO policy, only frames arriving between M_i^h and S_{max}^h on that node might accrue delay to the traversal time of f^m.

For an AFDX network in which all traffic has the same priority, the trajectory approach provides Equation (II.1) to compute an upper bound for the traversal time (R_i) encountered by a frame f^m of flow τ_i released at time t. (We present in Appendix B the TA equations for networks with multiple priority traffic.) $W_{i,t}^{last_i}$ represents the latest starting time of f^m on its destination node and C_i represents the transmission time of f^m.

$$R_i = \max\left(W_{i,t}^{last_i} + C_i - t\right) \tag{II.1}$$

The next equation shows the computation of $W_{i,t}^{last_i}$.

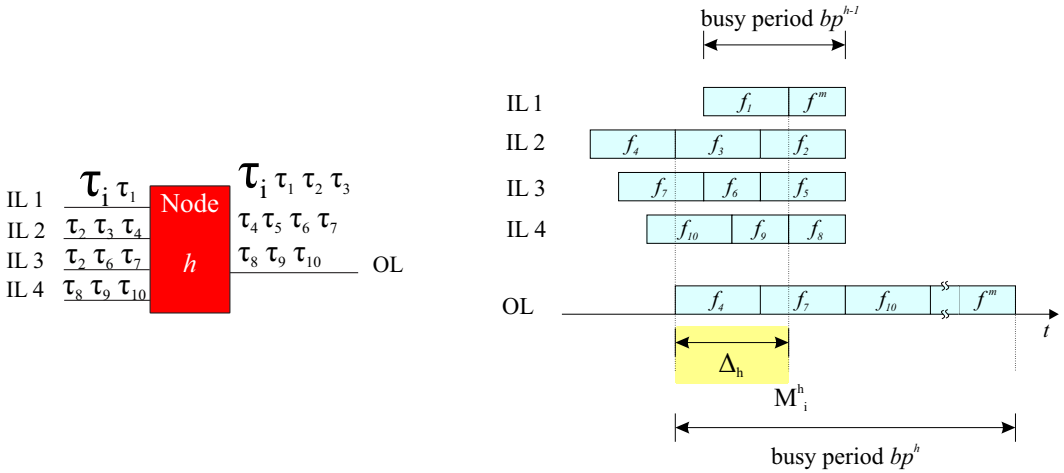
$$\begin{aligned} W_{i,t}^{last_i} = & \\ & \sum_{\substack{j\in\{1\ldots n\} \\ \mathcal{P}_j\cap\mathcal{P}_i\neq\emptyset}} \left(1 + \left\lfloor \frac{t + A_{i,j}}{T_j} \right\rfloor\right) \times C_j && \text{(II.2.1)} \\ & + \sum_{\substack{h\in\mathcal{P}_i \\ h\neq last_i}} \left(\max_{\substack{j\in\{1\ldots n\} \\ h\in\mathcal{P}_j}} \{C_j\}\right) && \text{(II.2.2)} \\ & + (|\mathcal{P}_i| - 1) \times sl && \text{(II.2.3)} \\ & + \sum_{\substack{h\in\mathcal{P}_i \\ h\neq first_i}} \Delta_h && \text{(II.2.4)} \\ & - C_i && \text{(II.2.5)} \end{aligned}$$

Where

$$\begin{aligned} A_{i,j} =& S_{max_i}^h - S_{min_j}^h - M_h^j + S_{max_j}^h, \\ T_j =& BAG_j, \end{aligned}$$

$\mathcal{P}_i$ represents the path of flow τ_i, sl represents the switching latency (called technological latency in [AFD09]) of a switch, and Δ_h represents the pessimism on node h due to the serialization effect (same effect as described in Section II.3.1.3). Note that Δ_h is not related to Δ presented in Chapter III.

Term (II.2.1) represents the delay accrual caused by frames on the busy period of f^m along its path from source to destination node; Term (II.2.2) accounts for the processing time of the largest frame for each node visited by f^m, i.e., the largest impact due to the store and forward property of the nodes; Term (II.2.3) represents the technological latency, i.e., time required by the switch fabric to relay the frame in all visited nodes;



(a) *Ten flows "meet" τ_i for the first time in node h*

(b) *Arrival and dispatching of frames on node h*

Figure II.10: *Serialization effect*

Term (II.2.4) accounts for the serialization effect on each visited node; and Term (II.2.5) represents the transmission time of f^m.

Term (II.2.4) has been introduced by Bauer et al. [BSF10] to eliminate the pessimism due to the serialization effect, from the original trajectory approach. Notice that Term (II.2.1) accounts for the frames of all flows that meet τ_i along its path. Term (II.2.1) assumes that, in the worst case, a frame of each crossing flow arrives at same time as f^m on the node where they first meet. This assumption is, however, pessimistic. Figure II.10a depicts an example of a node (h) in which ten flows ($\tau_1 \ldots \tau_{10}$) meet τ_i for the first time. Figure II.10b presents a possible arrival sequence for the frames of these ten flows. Notice that frames ingressing node h from the same input link (IL) cannot arrive on this node at the same time, i.e., they arrive *serialized*. Figure II.10b further shows that the data transmitted before M_i^h does not accrue any delay to the transmission time of f^m, and therefore, should be subtracted in the computation of $W_{i,t}^{last_i}$. Term (II.2.4) accounts for this subtraction on every traversed node. Bauer et al. show in [BSF10] that, for most of the analyzed cases, the trajectory approach with the serialization term leads to tighter end-to-end delay upper bounds, when compared to those computed by the original TA, network calculus or NC with serialization effect. Bauer et al. present in [BSF12c] a method to compute backlog upper bounds for AFDX buffers considering single priority traffic. The presented results show that buffer backlog upper bounds computed with TA can be up to 20% tighter than those computed with network calculus with serialization. The same authors extended the trajectory approach analysis for AFDX networks to compute end-to-end delay upper bounds for multiple priority traffic [BSF12b].

A later work by Kemayo et al. [KRR13] however, presents a counter example in which the computations with trajectory approach with serialization term leads to optimistic end-to-end delay upper bounds. This optimism occurs in some corner cases. The main reason for the optimistic results in some corner cases is the delay imposed to f^m caused

by frames arriving even before M_i^h. In some networks, the result of $(W_{i,t}^{last_i} + C_i - t)$ in Equation (II.1) is optimistic for some values of t. Nevertheless, this optimism does not appear at the result of Equation (II.1) if the serialization effect is not taken into account, i.e., Term (II.2.4) equals to zero. We refer the interested reader to a detailed example and respective explanations in [KRR13]. Li et al. present in [LCG14], a method to fix this optimism.

II.3.3 Forward End-to-End Delay Approach

Kemayo et al. present the forward end-to-end delay approach (FA) in [KRBR14]. Similar to the trajectory approach for single priority traffic, FA analyzes the busy period encountered by a frame of the flow under analysis. The main difference w.r.t. TA is that FA accounts for the impact of flows crossing τ_i by investigating the largest backlog generated by these flows. For details on the computations[1], we refer the interested reader to [KRBR14]. Simulation results of a small network show that, for flows that do not suffer from the serialization effect, the end-to-end delay upper bounds computed by FA are as tight as those computed by the trajectory approach and tighter than those computed by network calculus with and without serialization. For flows that suffer from serialization, FA leads to more pessimistic upper bounds when compared to those computed by TA and NC.

In [KBR+15], Kemayo et al. integrate the serialization effect into FA presenting a detailed analysis of the largest possible backlog accrued by frames ingressing the node from each input link individually. This analysis accounts for the serialization of frames on each input link. Results from the simulation of a small network show that the upper bounds computed by FA are very close to those computed by TA: some experiments show that FA leads to tighter bounds than TA, whereas in some other experiments, TA leads to tighter bounds. However, computations with FA for larger networks are not available. Consequently, the applicability of FA for large commercial airplanes still needs to be confirmed.

II.3.4 Simulation

As presented in Section II.1.3, AFDX nodes dispatch their frames asynchronously. Therefore, simulating all possible frame dispatch combinations is intractable for large airplane networks (the number of combinations explodes as the number of end-system and virtual links grows). Nevertheless, simulating the behavior of the network allows for the measurement of the exact end-to-end delay and buffer backlog of switches, and can be used to estimate the pessimism of upper bound computation methods. Considering that simulating a large network is intractable, Bauer et al. [BSF10] propose the simulation of the so called *unfavorable scenarios*. These scenarios are created by adjusting the release time of each frame on their source nodes, such that the combined transmission of all frames lead to the worst case end-to-end delay of a frame of the flow

[1]The interested reader should notice that in [KRBR14] and [KBR+15], the authors use a variable name t to represent the length of time interval under investigation, not a point in time.

under analysis. Obviously, since not all possible scenarios are simulated, the success of this method relies on the correct computation of the release time of frames, i.e., if the computed unfavorable scenario is not the worst case scenario, then this method leads to optimistic results.

II.3.5 Model Checking

This approach models the network as a timed automata and uses model checking to compute the exact worst case end-to-end delay encountered by frames traversing the network. Three main steps are required: i) model the network, ii) formalize the properties to check, and iii) run the model checker. One key aspect when computing upper bounds using model checking is the network model: a rather generalized model, which does not embed all properties of AFDX traffic, will be checked for a large number of non-valid states in the context of AFDX and demands larger computational efforts. Therefore, [ASEF10] also accounts for the serialization of frames, consequently reducing the search space and improving the performance. On one hand, despite the large improvement w.r.t. the computational demand, this approach does not scale and is therefore not used for the upper bound computations in large avionics networks. On the other hand, this approach can be used to evaluate the pessimism of the upper bounds achieved by other methods, e.g., trajectory approach, network calculus or forward end-to-end delay approach.

II.3.6 Holistic Approach

Gutiérrez et al. present in [GPGH14] a holistic approach to compute end-to-end delays on AFDX networks. This work takes into account the delay on the network starting at a higher level: the authors consider the time required for message fragmentation (into multiple frames) and the schedule of sub-virtual links. The comparison presented in [GPGH14] shows that the upper bounds computed by this approach for the investigated network configurations are more pessimistic than those of the trajectory approach (even without the serialization effect) and network calculus with the serialization effect.

II.4 Summary

In this chapter, we presented the fundamental properties of AFDX networks and the state-of-the-art approaches to compute upper bounds for end-to-end latency and buffer backlog. We showed in Section II.1 that AFDX networks can ensure predictable temporal properties for frames traversing the network by means of AFDX-compliant switches and end-systems. We described the concept of *virtual link* and showed how AFDX can provide bandwidth isolation. Section II.1 introduced the *contention* issue in AFDX networks, i.e., the condition when multiple frames try to egress an output port at the same time. AFDX addresses the contention issue by buffering, which in turn raises two important questions presented in Section II.2: What is the largest latency encountered by a frame due to buffering? And, what is the largest backlog in an output buffer?

In Section II.3 we presented the state-of-the-art methods to address these two questions. Due to the unsynchronized nature of AFDX networks and the large number of virtual links in large commercial airplane avionics systems, simulating all possible combinations of frames dispatching times is intractable. Nevertheless, we showed in Section II.3.4, that simulation is used in small networks to estimate the pessimism of other approaches. Therefore, a number of methods have been proposed to compute upper bounds for the worst case traversal time and largest buffer backlog. We divided the analysis of these existing methods into two categories: methods of the first category, analyze the traffic at flow level and bound the traffic and services on the network by means of *envelopes*; methods of the second analyze the network at a finer granularity by investigating the effects of the network service on a given frame. For most of the compared cases (see [BSF12b]), methods of the second category lead to the computation of tighter upper bounds when compared to those achieved by methods of the first category (including network calculus with serialization).

We showed that model checking has been used to compute the exact worst case end-to-end latency in AFDX networks. This method however, does not does not scale and is therefore not used for the upper bound computations in large avionics networks. Yet, this approach can be used to evaluate the pessimism of the upper bounds achieved by other methods.

Despite the importance of avoiding buffer overflows in AFDX networks, the majority of work on upper bound computation targets the worst case traversal time, not giving much attention to the buffer backlog analysis. We showed that, the state-of-the-art AFDX buffer backlog analysis does not provide a method to compute deterministic upper bounds for buffer backlog of AFDX networks with multiple priority traffic.

Chapter III

Buffer Backlog Upper Bound for AFDX Networks with Multiple Priority Traffic

This chapter presents a method to compute an upper bound for the backlog on each output buffer of AFDX networks with multiple priority traffic. In contrast to traditional methods that analyze the buffer backlog of multiple priority AFDX traffic at flow level (modeling network traffic and service with envelopes), our method analyzes the network at frame level.

We start by showing in Section III.1, that frames traversing an AFDX network face contention on the switches output ports. AFDX networks address this contention by means of *buffering*. Considering the safety-critical properties of avionics systems connected to AFDX networks, an overflow of these buffers and the resulting data loss must be avoided at all cost.

Section III.2 presents our method to compute a backlog upper bound for each output buffer on the network. This is the largest section in this chapter and is subdivided into eight subsections: Sections III.2.1, III.2.2, and III.2.3 present the terminology, assumptions and notations, respectively. We introduce our solution by presenting an overview in Section III.2.4.

Computing a buffer backlog upper bound depends on identifying the arrival of frames (competing frames) that leads to the largest buffer backlog. Notice that not only the amount and sizes of these competing frames should be accounted for. Also the order and the points in time in which they ingress in the output buffers impact the buffer backlog. We introduce the concept of *intervals* in Section III.2.5 and present, in Section III.2.6, the properties that define the arrival of competing frames, in each type of *interval*, leading to the largest buffer backlog. Section III.2.7 presents the formulas to compute an upper bound for this buffer backlog. In Section III.2.8 we discuss how the results achieved by our method can be applied to AFDX switches that do not conform with our assumptions on how the ingress and egress of frames are handled by AFDX switches. Section III.3 presents the computational costs of our method when implemented in two different programming languages. Section III.4 concludes this chapter with a summary.

III.1 Introduction

An increasing number of modern aircraft, adopts Avionics Full DupleX switched Ethernet (AFDX) for data communication of safety-critical avionics systems, replacing formerly deployed point-to-point networks. In order to ensure the correct timing behavior of these safety-critical systems, AFDX networks must ensure two important properties: no (or minimal) frame loss and bounded end-to-end latency (Section II.2). As presented in Section II.1, AFDX is based on switched Ethernet and therefore no data loss or frame traversing delay occurs due to collision. Contention however, frequently occurs on the output ports of AFDX switches increasing frames latency eventually leading to data loss.

ARINC 667 Part 7 does not specify any means for node synchronization, i.e. data communication between connected nodes occurs asynchronously. One positive effect of this property is that AFDX nodes do not need to maintain a clock synchronization protocol, which on occurrence of failures, could lead to data losses. However, due to the unsynchronized data transmission, data contention occurs on switches output ports whenever frames from different physical links ingress AFDX switches and try to egress from the same port at the same moment. ARINC 664 Part 7 [AFD09] addresses the contention issue by means of buffering, i.e., AFDX switches implement buffers on their output ports to store contained frames. Most avionics systems connected via AFDX networks are considered safety-critical. Therefore, buffer overflow and the resulting data loss in AFDX networks must be avoided at all cost. In order to achieve this goal, the network designer must specify the memory reserved for each buffer of every output port of all switches on the network such that no buffer overflow occurs. At the same time, over-reservation should be avoided.

The AFDX standard defines *two* priority levels to classify virtual links. Yet, existing commercial AFDX solutions allow for the classification of virtual links into *multiple* priority levels. For instance, the AFDX switch presented in [TTT] permits the classification of virtual links with eight priorities. Considering that up to eight values of bandwidth allocation gap (BAG) are allowed in the current ARINC 664 Part 7 standard, if priorities are assigned according to BAGs (higher priorities to virtual links with smaller values of BAG), this commercial solution permits the generation of a non-preemptive rate monotonic schedule of frames. In order to provide a buffer analysis for AFDX networks that provide multiple priority traffic (obviously, accounting for the current two priority AFDX standard), we assume in this chapter that virtual links can be classified into one out of many (not only two) priority levels.

Computing the exact largest value for the buffer backlog of each buffer is intractable for an industrial size AFDX network since the number of possible combinations of arrival orders of frames on an output port grows factorially with the number of frames in the busy period[1]. Therefore, we present in this chapter a method to compute an upper

[1] In this dissertation, we consider the busy period definition presented in [DBBL07]. The main property of this definition is that, in a busy period of priority P (bp^P), all frames of priority P or higher ready for transmission strictly before the end of bp^P are transmitted in this busy period, and therefore do not impact frames that do not belong to bp^P.

bound for those buffer backlogs. Further, we present the properties of the scenario leading to the largest buffer backlog of each output buffer in an AFDX network and the formulas to compute an upper bound for these buffer backlogs. The results presented in this chapter allow for the AFDX network designer to safely allocate memory to each output buffer in an AFDX network with multiple priority virtual links.

III.2 Upper Bound Computation

The computation of an upper bound for the backlog encountered by a frame, e.g., f^m, entering an output buffer on an AFDX network depends on the identification of:

- the frames that form the worst case busy period encountered by f^m on the output port under analysis
- the arrival order of these frames (frames in the worst case busy period) that lead to the largest buffer backlog

The buffer backlog upper bound computation presented in this chapter assumes that the frames in the busy period are given, i.e., computed by one of the methods presented in Chapter II. In Appendix B we present how one of these methods (trajectory approach) can be used to compute the competing frames.

III.2.1 Terminology

In this dissertation, we use the terms *time of transmission* and *time of arrival* to refer to the time when a frame is completely transmitted and received, respectively. Further, the terms *ingress* and *arrival of frames*, and *egress* and *transmission of frames* are used interchangeably. We use the term *competing frames* to refer to all frames present in the busy period of the virtual link under analysis. *Scenario* refers to a specific ingress order and the respective egress order of competing frames. Finally, whenever not mentioned otherwise, we use *buffer* to refer to a switch output port buffer. Notice that we assume that on each switch output port, there exists one buffer for each priority level (see Section III.2.2).

III.2.2 Assumptions

We assume an AFDX network $\mathcal{N}$ with the following characteristics:

- all nodes (end-systems and switches) comply with the ARINC 664 Part 7 standard
- the network topology is static and defined off-line
- the connection between two nodes is done via two unidirectional physical links
- the maximum bandwidth (BW) is constant and has the same value on every physical link, e.g. 100Mbps.

- a set of virtual links, considering that each virtual link is defined off-line with the following properties:
  - maximum frame size (S_{max})
  - bandwidth allocation gap (BAG)
  - maximum jitter (J_{max})
  - fixed physical path (route) from source to destination(s) end-systems
  - one priority level P, out of a set of priority levels $\mathcal{P}$, s.t. $\mathcal{P} = \{1, 2, 3, \ldots, n\}$
- a set of switches with the following properties:
  - every output port of each switch has one buffer for each priority level
  - frames are dispatched according to the store and forward paradigm, i.e. a frame can only egress after it is completely stored in the output buffer
  - stored frames are dispatched to the output port according to the Fixed Priority First-In First-Out (FP/FIFO) policy
  - frames are copied to and removed from the output buffers bit by bit as frames ingress and egress the switch, respectively

We consider that a switch *output port* is composed of a frame dispatcher, output buffers (one buffer per priority level on each output port), and an output link (OL).

III.2.3 Notations

Table III.1 presents the terms, indices and sets used to describe and to compute the upper bound for the buffer backlog in each buffer on an AFDX network.

Table III.1: *Notations*

$\mathcal{N}$	AFDX network, i.e., switches, end-systems, network topology and virtual links (including their routes)
BW	Network bandwidth
$\mathcal{O}$	Set of output ports in the network
o, P, i, v	Indices used to represent a switch output port, priority, input link and a virtual link, respectively
$\mathcal{CF}^{v,o}$	Set of all competing frames in the largest busy period encountered by a frame of the virtual link v on the output port o
$\mathcal{P}^o$	Set of priorities of all *VL*s that egress the output port o
$\mathcal{I}^o$	Set of input links, connected to the same switch as the output port o, from which the competing frames ingress
$\mathcal{V}^{o,P,i}$	Set of virtual links that egress the output port o with priority P and ingress from the input link i
P-buffer	Buffer of priority P
P-frame	Frame of priority P
P-data	Data of priority P

f^{max}	Largest frame among all competing frames
$f^{LP,max}$	Largest competing frame with priority lower than P
$*$	Index used to represent the scenario leading to the largest backlog faced by a virtual link (worst case scenario)
$\curlywedge$	Index used to represent a scenario other than the worst case scenario
ω	Point in time when the first competing frame starts to ingress
α	Point in time when the first competing frame starts to egress
β	Point in time when the last competing frame of priority other than the frame under analysis completely egresses
θ^P	Point in time when the last competing frame of the priority P or higher ingresses
$s(data)$	Function that computes the size of $data$
$t(size)$	Function that computes the time required to transmit data of length $size$ on a physical link, i.e., $t(size) = size/BW$
$d(time)$	Function that computes the maximum amount of egress data on any physical link during a time span $time$, i.e., $d(time) = time * BW$
σ^P_{ALL}	Sum of the size of all competing P-frames
$\sigma^P_{Interval}$	Sum of the size of competing P-data that egresses during a given interval
$R^{*P}_{(b-a)}$	Amount of ingress P-data in scenario * during the interval $]a, b]$, i.e., $R^{*P}_{(b-a)} = R^{*P}_b - R^{*P}_a$, s.t. $R^{*P}_a = R^{*P}_{(a-0)}, R^{*P}_b = R^{*P}_{(b-0)}$
$T^{*P}_{(b-a)}$	Amount of egress P-data in scenario * during the interval $]a, b]$, i.e., $T^{*P}_{(b-a)} = T^{*P}_b - T^{*P}_a$, s.t. $T^{*P}_a = T^{*P}_{(a-0)}, T^{*P}_b = T^{*P}_{(b-0)}$
$B^{*P}_{\theta^{*P}}$	P-buffer backlog at time θ^{*P} in the worst case scenario
$B^{o,P,v,max}$	Largest backlog of a buffer in node o of priority P perceived by a frame of virtual link v
$B^{o,P,max}$	Largest backlog of a buffer in node o of priority P perceived by *any* frame of priority P
$bl^P(\tau)$	Function to compute P-buffer backlog at time t
$\delta R^{\curlywedge P}_\tau$	Difference between the amount of P-data that ingresses during the time interval $]\omega^*, \tau]$ in the scenario $\curlywedge$ and *
$\delta T^{\curlywedge P}_\tau$	Difference between the amount of P-data that egresses during the time interval $]\omega^*, \tau]$ in the scenario $\curlywedge$ and *

Notice that Table III.1 shows three functions that address the computation of frame properties. For a given frame f^a:

- $s(f^a)$ computes the size of f^a
- $t(s(f^a))$ computes the time required to transmit f^a
- $d(t(s(f^a)))$ computes the amount of egress data during the interval of time in which f^a egresses. The result of this equation is equivalent to $s(f^a)$, i.e., $d(time)$

is the inverse function of $t(size)$. Notice that, while $s(size)$ takes a *frame* as input, $d(time)$ takes a time value as an input.

We refer to the positive part of any variable with the notation $()^+$, e.g. $(\Delta)^+ = \max(0, \Delta)$. Further, we use the symbols $\bar{i}, \bar{v}, \bar{P}$, respectively, to represent input links, virtual links and priorities other than those already used in the outer loop described in Algorithm 1.

III.2.4 Method Overview

When computing the backlog of any buffer on an AFDX network, we do not to consider the complete network at once. Rather, for the backlog upper bound computation of a buffer, we only investigate the network traffic that impacts the busy period of the virtual links that egress from the same output port as the buffer under analysis. Once the set of competing frames is known, we only investigate the respective output port, the buffer under analysis and the physical links where the competing frames ingress.

Figures III.1, III.2 and III.3 show, based on an example, the relationship between the complete AFDX network and the buffer under analysis. In this example, we analyze the backlog of the buffer connected to the output port 9 of switch 2 (connecting switch *S2* to switch *S3*) in Figure III.1. Notice that each line connecting two entities in this figure is a bi-directional link. We consider in this chapter that, each bi-directional link is modeled by two unidirectional links. We assume that all traffic that impacts the computation of the busy period under analysis reaches the output port of *S2* from end-systems and switches as depicted by the green arrows in Figure III.2. Once the set of competing frames is known, the analysis can be restricted to the entities depicted in Figure III.3 and presented next:

- *Switch output port* (o) which the buffer under analysis is part of.
- *Buffer under analysis* (only the buffer under analysis needs to be considered).
- *Virtual links* that egress from output port o. Notice that only frames of these virtual links are present in the busy period under analysis. These frames are called competing frames ($\mathcal{CF}^{v,o}$).
- *Input links* from where competing frames ingress. Input links IL1, IL2, IL3, and IL4 in Figure III.3, represent the links connecting the three end-systems e_7, e_8, e_9 and the switch *S1* to switch *S2*.

Algorithm 1 presents an overview of the upper bound computation for the buffer backlog on the output ports of AFDX nodes. To compute an upper bound for the backlog of a buffer on the network (for each output port o, and each buffer of priority P– Lines 1 and Line 2 of Algorithm 1), we execute six basic steps:

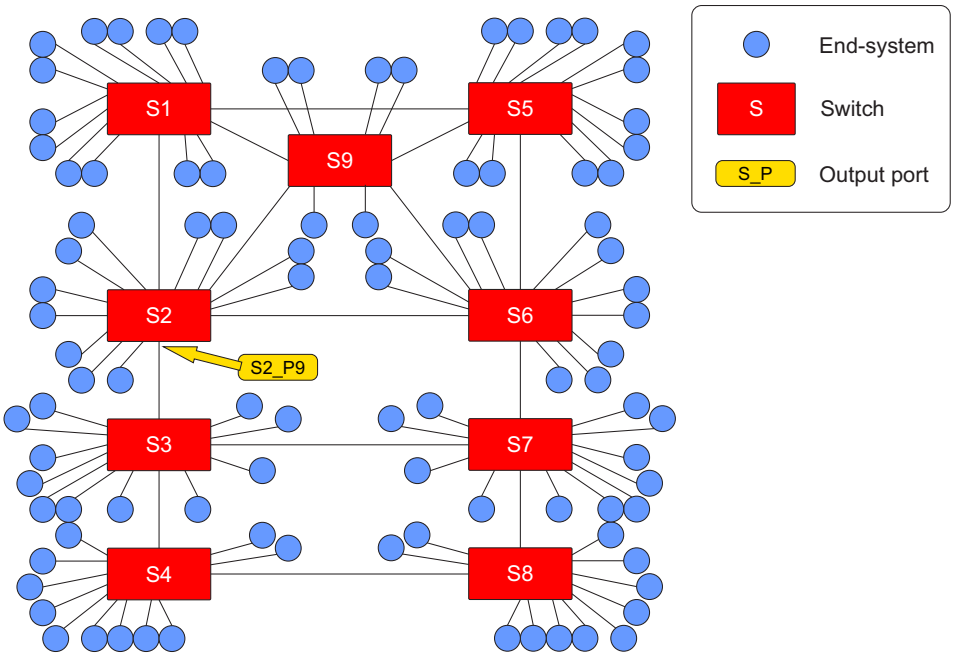


Figure III.1: *Publicly available AFDX topology of the Airbus A380. Output port under analysis is port 9 of switch 2 (S2_P9).*

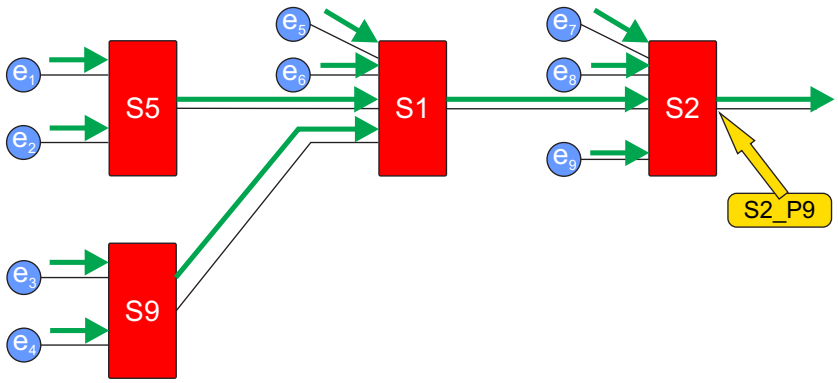


Figure III.2: *Traffic that impacts the computation of the busy period under analysis marked with green lines*

i) identify all virtual links that egress from the same output port as the buffer under analysis with the same priority as this buffer, i.e. generate the set $\mathcal{V}^{o,P}$.

ii) select each *virtual link* (v) with the same priority (P) as the buffer under analysis – Line 3

iii) compute the *frames in the largest busy period* encountered by a frame of virtual link v, i.e. the competing frames ($\mathcal{CF}^{v,o}$) – Line 4

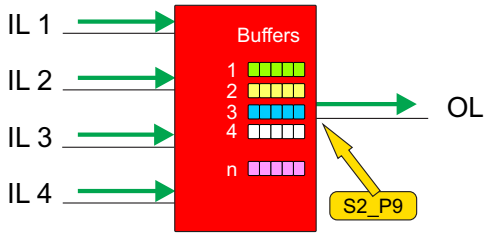


Figure III.3: *AFDX output port: buffers (one per priority) and output link. Further, the input links from where competing frames ingress the switch. Green lines represent ingress and egress competing frames*

iv) determine the properties of the worst case scenario – part of Line 5[2]

v) compute the *largest backlog encountered by a frame of virtual link* v ($B^{o,P,v,max}$) – Line 5

vi) compute an *upper bound for the buffer backlog* ($B^{o,P,max}$) by calculating the maximum largest backlog among all virtual links with priority P that egress the output port o – Line 6

Algorithm 1 Upper bound computation overview

1: **for each** output port $o \in \mathcal{O}$ **do**
2: **for each** priority $P \in \mathcal{P}^o$ **do**
3: **for each** virtual link $v \in \mathcal{V}^{o,P}$ **do**
4: $\mathcal{CF}^{v,o} = \text{GetCompetingFrames}(\mathcal{N}, o, v)$
5: $B^{o,P,v,max} = \text{ComputeBacklog}(\mathcal{N}, \mathcal{CF}^{v,o}, P)$
6: $B^{o,P,max} = \max_{\forall v \in \mathcal{V}^{o,P}} (B^{o,P,v,max})$

In order to improve readability, the next sections omit the output port index o whenever possible.

III.2.5 Intervals

We define four types of time intervals to facilitate the buffer backlog analysis. According to the ingress and egress of frames with same priority as the buffer under analysis, each of these intervals present a specific backlog accrual. For the backlog analysis of a buffer of a given priority P, Table III.2 presents the buffer backlog accrual according to the ingress and egress of P-frames and the size of the respective interval $s(Interval)$. It is noteworthy that, when analyzing the ingress and egress of P-frames, any time interval that does not match the properties of Table III.2 can be decomposed into smaller intervals until each of them adheres to properties presented in that table.

[2]The theoretical analysis of the worst case scenario is presented in detail in Section III.2.6 and need not be done in every iteration of Algorithm 1; only the computations resulting from this analysis are executed in Line 5 of Algorithm 1.

Table III.2: *Interval types description and respective buffer backlog accrual*

interval type	ingress of P-frames	egress of P-frames	backlog accrual
1	Yes	No	$\sigma^P_{Interval}$
2	Yes	Yes	$\sigma^P_{Interval} - s(Interval)$
3	No	No	0
4	No	Yes	$-s(Interval)$

Figure III.4 depicts the ingress and egress of frames on a node and the respective classification of intervals. Notice that frames are classified into three priority groups: P, HP and LP, respectively for the priority under analysis, higher and lower priorities than P. The upper part of the figure presents the competing frames ingressing from the input links IL1 to IL4 and the egress frames from the output link (OL). The central part of Figure III.4 presents the backlog on P-buffer and the lower part shows the interval types.

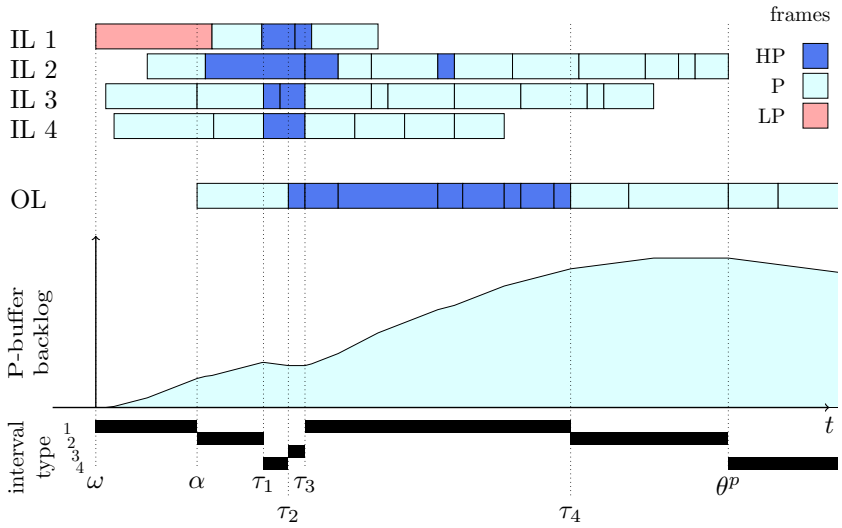


Figure III.4: *Example scenario. Ingress and egress frames at the top, buffer backlog in the middle and interval types at the bottom*

In intervals of type 1, P-frames ingress and no P-frame egresses during the whole interval (see time intervals between ω and α, and τ_3 and τ_4). Consequently, the backlog accrual is equal to the sum of all P-data that ingresses the P-buffer in these intervals. In intervals of type 2, P-frames ingress and egress during the whole interval (see time intervals between α and τ_1, and τ_4 and θ^P). Consequently, the backlog accrual is equal to the sum of all P-data that ingresses the P-buffer minus the length of the interval. In intervals of type 3, no P-frame ingresses or egresses the buffer. Consequently, the

P-buffer backlog does not change (see time interval between τ_2 and τ_3). In intervals of type 4, no P-frame ingresses while P-frames egress during the whole interval (see time intervals between τ_1 and τ_2, and after θ^P). Thus, the backlog decreases during type 4 intervals.

In every scenario, four events define important points in time for the analysis of the worst case buffer backlog: ω, α, β and θ^P (see Figures III.4, III.5 and III.6). Notice that, in the next paragraphs, we omit the scenario index (*, $\curlywedge$) in the notations of these points in time since the actual explanations apply to any scenario.

- ω: point in time when the first competing frame starts to ingress
- α: point in time when the first competing frame starts to egress
- β: point in time when the last competing frame of priority other than the frame under analysis completely egresses
- θ^P: point in time when the last competing frame of the priority P or higher ingresses

These points in time mark the start or end of some of the intervals described in Table III.2. For the sake of simplicity and without loss of generality, we assume in this chapter that the start of arrival of the first ingress frame occurs at $t = 0$, i.e. $\omega = 0$. Additionally, in order to facilitate the presentation of the effects of different scenarios on the buffer backlog accrual, we define a new variable Δ to represent the amount of egress data between β and θ^P, i.e. $\Delta = d(\theta^P - \beta)$. Negative values of Δ reflect scenarios in which the egress of the last frame of priority other than P occurs after the ingress of the last P-frame, i.e. $\beta > \theta^P$, which are analyzed differently. Figure III.5 presents such a scenario. Notice that Δ is not related to Δ_h presented in Chapter II.

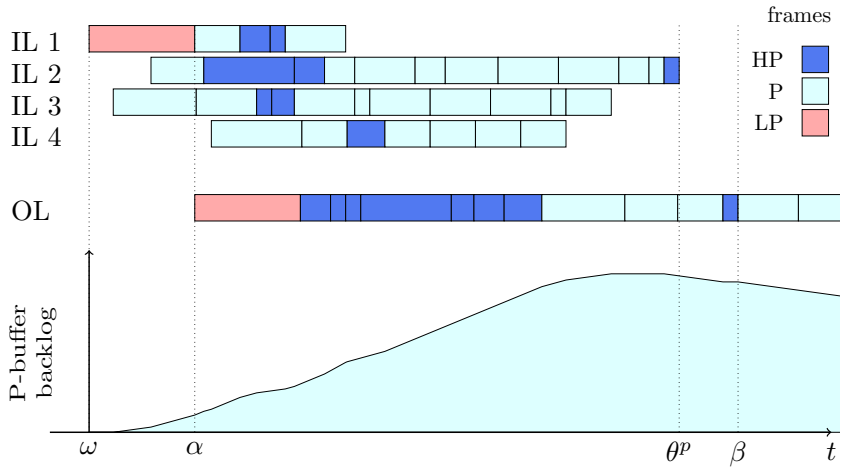


Figure III.5: *Example scenario with* $\beta > \theta^P$

III.2.6 Worst Case Scenario

We define *worst case scenario* as the ingress order of competing frames (and consequent egress order) that leads to the largest backlog of the buffer under analysis encountered by a frame, e.g., f^m, of the virtual link v. Figure III.6 depicts the worst case scenario (represented with the index *) for the P-buffer backlog considering the same set of competing frames as in Figure III.4. Theorem III.1 describes the properties of the worst case scenario for the P-buffer backlog encountered by a frame of the virtual link v (of priority P).

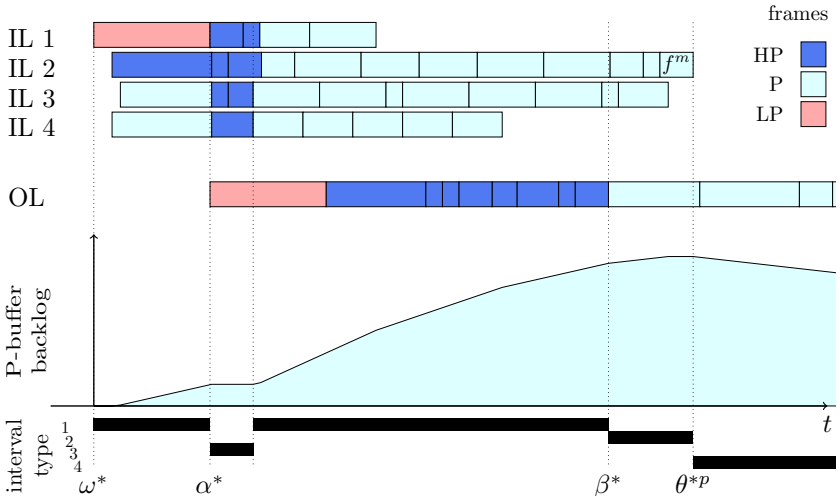


Figure III.6: *Worst-case scenario for P-buffer considering the same competing frames as in Figure III.4*

Theorem III.1. *The scenario leading to the largest backlog encountered by a frame f^m of a virtual link of priority P presents the following properties: i) the smallest sum of lengths of type 2 intervals, ii) no type 4 interval before the arrival of the last P-frame, i.e., not before θ^{*P}, iii) no type 1 or type 3 interval occurs after a type 2 interval, iv) the largest P-buffer backlog occurs at the point in time when the last P-frame ingresses, i.e., at θ^{*P}, v) f^m is the last ingressing competing frame.*

Next, we prove the five properties of Theorem III.1.

i) **Smallest sum of lengths of type 2 intervals**

Proof. At any moment, the backlog of P-buffer, is equal to the amount of ingress data minus the amount of egress data until that moment. Notice that θ^{*P} is the point in time when the last P-frame ingresses. Considering that, in the worst case scenario, no type 4 interval occurs until θ^{*P}, the backlog ($B^{*P}_{\theta^{*P}}$) at this point in time is equal to the sum of all ingress P-data minus the sum of all egress P-data, during type 2 intervals, until θ^{*P}. Equation III.1 presents this relationship:

$$B^{*P}_{\theta^{*P}} = \sigma^{P}_{ALL} - \sum s(Interval_{type\_2}) \tag{III.1}$$

Consequently, the smaller the sum of lengths of type 2 intervals, the larger the P-buffer backlog at θ^{*P}. Notice that property iv) of Theorem III.1 states that the largest backlog of P-buffer occurs at θ^{*P}. Thus, the worst case scenario contains the smallest sum of lengths of type 2 intervals until θ^{*P}. ■

ii) **No type 4 interval occurs before the arrival of the last P-frame**

Proof. During intervals of type 4, the amount of data in the P-buffer decreases (see Section III.2.5). Further, the occurrence of a type 4 interval does not favor the occurrence of any other interval that compensates the decrease of the buffer backlog w.r.t. the worst case scenario (caused by this type 4 interval).

Let us compare the buffer backlog of a scenario ' in which a type 4 interval of size $s(I_4)$ occurs from τ until $\tau + s(I_4)$ before θ^{*P}, against the worst case scenario. At the end of this type 4 interval, the buffer backlog in scenario ↛ is smaller than in the worst case scenario (Table III.2 shows that type 4 is the only interval type with negative backlog accrual). The difference on buffer backlog accrual at the end of this type 4 interval depends on the type of interval during $[\tau, \tau + s(I_4)]$ in the worst case scenario. Table III.3 presents the accrual difference w.r.t. all types of interval in the worst case scenario.

Table III.3: *Difference on the backlog accrual caused by replacing a type 1, 2 and 3 with a type 4 interval*

interval type replacement	backlog accrual difference
1 with 4	$-(\sigma_{I_4}^{*P} + s(I_4))$
2 with 4	$-\sigma_{I_4}^{*P}$
3 with 4	$-s(I_4)$

Notice that, for a type 1 interval (I_4) to occur, no P-frame may ingress during $[\tau, \tau + s(I_4)]$. Therefore, the amount of P-data stored in P-buffer until $\tau + s(I_4)$ decreases by $\sigma_{I_4}^{*P}$ (the amount of P-data stored between $[\tau, \tau + s(I_4)]$ in the worst case scenario). Consequently, $\sigma_{I_4}^{*P} = 0$ is the largest possible increase on the backlog accrual after the interval I_4. Recall that $\sigma_{I_4}^{*P} = 0$ for type 3 intervals. Thus, no increase in the backlog accrual is possible by replacing a type 3 with a type 4 interval. Replacing intervals of type 1 with type 4 and type 2 with type 4 leads to decreasing the backlog by $(\sigma_{I_4}^{*P} + s(I_4))$ and $\sigma_{I_4}^{*P}$, respectively. These two values are larger than or equal to the largest possible backlog accrual after the interval I_4, i.e., larger than $\sigma_{I_4}^{*P}$. Hence, the occurrence of type 4, and the resulting replacement with one of the other interval types, before θ^{*P} does not lead to a scenario in which the backlog in P-buffer is larger than the largest backlog in the scenario described by Theorem III.1. ■

iii) **No type 1 or type 3 interval occurs after a type 2 interval**

Proof. Let us consider a scenario $\curlywedge$ (different from the worst case scenario as described by Theorem III.1) in which a type 1 or type 3 occurs after a type 2 interval. This implies that, in this scenario $\curlywedge$, P-frames egress before $\beta^{\curlywedge}$, where $\beta^{\curlywedge}$ represents the transmission time of the last frame of priority other than P in scenario $\curlywedge$, i.e., the counterpart of β^*. Next, we compare the buffer backlog in scenario $\curlywedge$ against the backlog in the worst case scenario, and prove that there exists no scenario in which the buffer backlog is larger than the backlog at θ^{*P} in the scenario described by Theorem III.1.

Let $B_{\tau}^{\curlywedge P}$ be the P-buffer backlog of scenario $\curlywedge$ at a given point in time τ and B_{τ}^{*P} be the buffer backlog of scenario * at the same point in time. Next, we prove that, at any point in time between $\omega^{\curlywedge}$ and $\theta^{\curlywedge P}$, the P-buffer backlog in scenario $\curlywedge$is less than or equal to the P-buffer backlog at θ^{*P} in the worst case scenario. Notice that property iv) of Theorem III.1 proves that the P-buffer backlog does not increase after the arrival of the last P-frame (θ^{*P}). Thus, our claim holds if:

$$B_{\theta^{*P}}^{*P} - B_{\tau}^{\curlywedge P} \geq 0 \ \forall \ \omega^{\curlywedge} \leq \tau \leq \theta^{\curlywedge P}$$

The buffer backlog of scenario $\curlywedge$ at any point in time τ is equal to the buffer backlog of scenario * at the same τ plus the difference (increase or decrease) on the amount of data that ingresses ($\delta R_{\tau}^{\curlywedge P}$), minus the difference on the amount of data that egresses ($\delta T_{\tau}^{\curlywedge P}$) the P-buffer in the time interval $]\omega, \tau]$. Assuming $\omega^* = \omega^{\curlywedge} = 0$, we represent $B_{\tau}^{\curlywedge P}$ as:

$$B_{\tau}^{\curlywedge P} = B_{\tau}^{*P} + \delta R_{\tau}^{\curlywedge P} - \delta T_{\tau}^{\curlywedge P} \tag{III.2}$$

where

$$\delta R_{\tau}^{\curlywedge P} = R_{\tau}^{\curlywedge P} - R_{\tau}^{*P} \tag{III.3}$$

$$\delta T_{\tau}^{\curlywedge P} = T_{\tau}^{\curlywedge P} - T_{\tau}^{*P} \tag{III.4}$$

Figure III.7 illustrates the properties of $\delta R_{\tau}^{\curlywedge P}$ and $\delta T_{\tau}^{\curlywedge P}$. In this example, the value of $\delta R_{\tau}^{\curlywedge P}$ and $\delta T_{\tau}^{\curlywedge P}$ at $t = \tau_1$ are respectively $s(f_2'^1)$ (size of the first ingress frame in input link 2 of scenario $\curlywedge$) and $\tau_1 - \alpha^{\curlywedge}$. At $t = \tau_2$, $\delta R_{\tau}^{\curlywedge P}$ is equal to 0 since the same amount of P-data ingresses in both scenarios. The value of $\delta T_{\tau}^{\curlywedge P}$ is equal to $s(f_2'^1)$.

In the worst case scenario, we represent the backlog at θ^{*P} w.r.t. any point in time τ as:

$$B_{\theta^{*P}}^{*P} = B_{\tau}^{*P} + R_{(\theta^{*P}-\tau)}^{*P} - T_{(\theta^{*P}-\tau)}^{*P} \tag{III.5}$$

Consequently, from Equation (III.2) and (III.5) we represent the difference between the buffer backlog of the worst case scenario at θ^{*P} and of scenario $\curlywedge$ at time τ as:

$$B_{\theta^{*P}}^{*P} - B_{\tau}^{\curlywedge P} = R_{(\theta^{*P}-\tau)}^{*P} - T_{(\theta^{*P}-\tau)}^{*P} - \delta R_{\tau}^{\curlywedge P} + \delta T_{\tau}^{\curlywedge P} \tag{III.6}$$

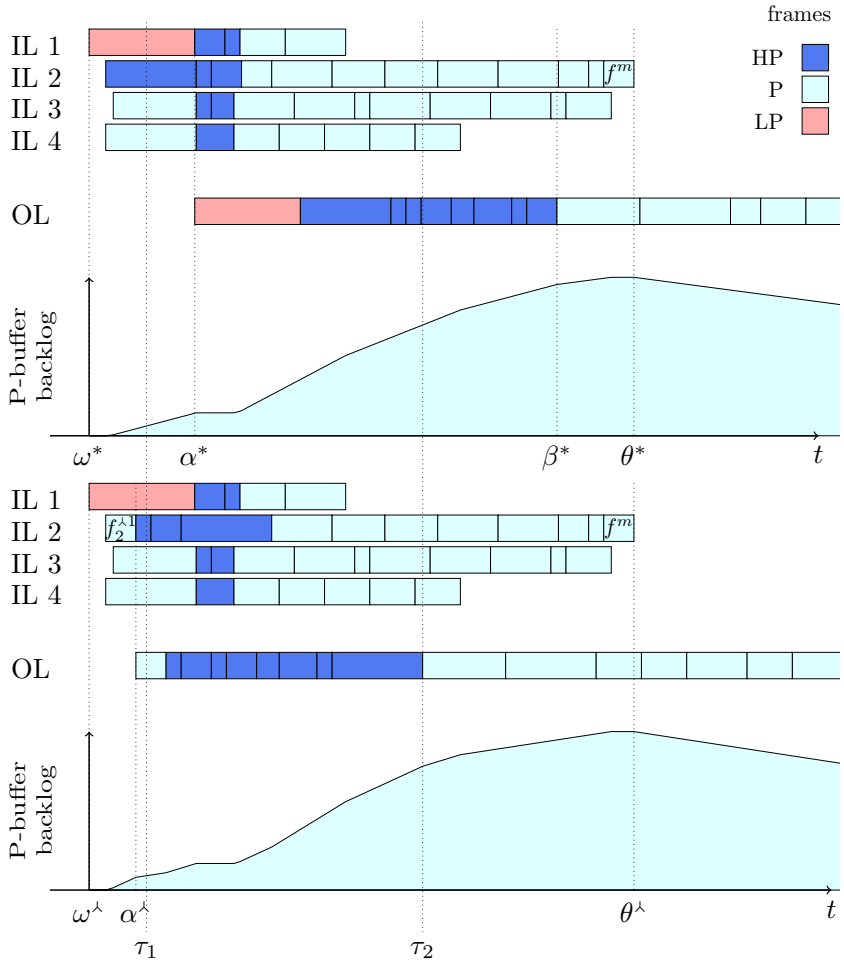


Figure III.7: *Scenarios to illustrate the computation of $\delta R_\tau^{\curlywedge P}$ and $\delta T_\tau^{\curlywedge P}$. On top, the worst case scenario*

Per definition, $R_{(a-b)} = R_a - R_b$. Then:

$$R^{*P}_{(\theta^{*P}-\tau)} = R^{*P}_{\theta^{*P}} - R^{*P}_{\tau} = \sigma^{P}_{ALL} - R^{*P}_{\tau} \tag{III.7}$$

and:

$$\sigma^{P}_{ALL} = R^{\curlywedge P}_{\tau} + R^{\curlywedge P}_{(\theta^{\curlywedge P}-\tau)} \tag{III.8}$$

Substituting (III.8) into (III.7) and then into (III.6):

$$\begin{aligned} B^{*P}_{\theta^{*P}} - B^{\curlywedge P}_{\tau} =& R^{\curlywedge P}_{\tau} + R^{\curlywedge P}_{(\theta^{\curlywedge P}-\tau)} - R^{*P}_{\tau} - T^{*P}_{(\theta^{*P}-\tau)} - \delta R^{\curlywedge P}_{\tau} + \delta T^{\curlywedge P}_{\tau} \\ =& R^{\curlywedge P}_{\tau} + R^{\curlywedge P}_{(\theta^{\curlywedge P}-\tau)} - R^{*P}_{\tau} - T^{*P}_{(\theta^{*P}-\tau)} - (R^{\curlywedge P}_{\tau} - R^{*P}_{\tau}) + \delta T^{\curlywedge P}_{\tau} \end{aligned}$$

we have:

$$B^{*P}_{\theta^{*P}} - B^{\curlywedge P}_{\tau} = R^{\curlywedge P}_{(\theta^{\curlywedge P}-\tau)} - (T^{*P}_{(\theta^{*P}-\tau)} - \delta T^{\curlywedge P}_{\tau}) \tag{III.9}$$

We prove in Appendix (A.2.2) that the value computed by Equation III.9 is larger than or equal to zero for any value of τ between $\omega^{\curlywedge}$ and $\theta^{\curlywedge P}$, i.e., the P-buffer

backlog in scenario λ is less than or equal to the P-buffer backlog at θ^{*P} in the worst case scenario. . Notice that we do not investigate the P-buffer backlog for values of τ larger than $\theta^{\lambda P}$ since no P-frame ingresses after this point in time, and consequently the respective buffer backlog does not increase. ■

iv) **The largest P-buffer backlog occurs at θ^{*P}**

Proof. Let us define a function $bl^P(\tau)$, where t represents time, to express the P-buffer backlog. Considering that the backlog of P-buffer does not increase after the arrival of the last P-frame, $bl^P(\tau)$ is a decreasing function for values of τ larger than $\theta^{\lambda P}$. Thus:

$$bl^P(\tau) \leq bl^P(\theta^{*P}), \forall\ \tau \geq \theta^{*P}$$

According to Theorem III.1, no type 4 interval occurs before θ^{*P} in the worst case scenario, i.e. the backlog on the P-buffer does not decrease until θ^{*P}. Therefore, $bl^P(\tau)$ is an increasing function for values of t between ω^* until θ^{*P}, i.e.:

$$bl^P(\tau) \leq bl^P(\theta^{*P}), \forall\ \omega^* \leq \tau \leq \theta^{*P}$$

Consequently, at θ^{*P}, the P-buffer has the largest backlog. ■

v) **f^m is the last ingressing competing frame**

Proof. The proof of Property iv) of Theorem III.1 shows that for a given set of competing frames, the largest backlog of the P-buffer occurs at θ^{*P}. Considering that this theorem provides the worst case scenario properties, i.e., the properties of the scenario leading to the largest backlog encountered by a frame (f^m) of the virtual link v, then f^m must be the last ingress frame, i.e. ingress at θ^{*P}. ■

III.2.6.1 Computation of $\beta^*, \alpha^*, \theta^{*P}$

This section presents the computation of the three important points in time – α, β and θ^P – used to identify the worst case scenario and to compute the buffer backlog upper bound. The next two statements provide important properties to identify the worst case scenario:

1. **From ω^* until θ^{*P}, P-frames might egress only in type 2 intervals.**
   Per definition, P-frames might only egress in intervals of type 2 or 4 (see Table III.2). Further, in the worst case scenario (according to Theorem III.1), no type 4 occurs before θ^{*P}.

2. **There exists only one type 2 interval in the worst case scenario: the interval $[\beta^*, \theta^{*P}]$.**
   According to Theorem III.1, no type 1 or type 3 interval occurs after a type 2 interval. Remember that no interval of type 4 occurs until θ^{*P}, i.e., P-frames egress only in type 2 intervals until θ^{*P}. Therefore, the start of a type 2 interval

defines β^*, the moment in time when the last frame of priority other than P egresses. Further, since no type 4 interval occurs until θ^{*P}, the interval $[\beta^*, \theta^{*P}]$ is a type 2 interval, and no other type 2 interval exists.

Consequently, the smallest sum of lengths of type 2 intervals, which is in fact only one type 2 interval, requires *the largest* β^* and *the smallest* θ^{*P}.

III.2.6.1.1 Computation of β^* Notice that the largest β^* implies the largest amount of time, after ω^*, in which no P-frame egresses. Three effects may delay the transmission of the first P-frame, thus defining the point in time β^*: *idle time, blocking* and *interference*. Next we address each of these effects.

The store and forward characteristic of AFDX switches intrinsically leads to *idle time* during the arrival of the first competing frame which, in turn, delays the transmission of all competing frames. *Blocking* and *interference* occur when the delay on the transmission of a frame is caused by the transmission of frames of lower (LP-frames) and higher (HP-frames) priority, respectively. In the worst case scenario, the largest sum of those effects occurs and results in the value for β^*.

$$\beta^* = idle_v^* + interf_v^* + block_v^* \tag{III.10}$$

where $idle_v^*$, $interf_v^*$, $block_v^*$ represent the worst case scenario idle, interference and blocking time for any possible scenarios of a set of competing frames, respectively.

III.2.6.1.1.1 Worst case idle time and α^* Store and forward switches used in AFDX networks intrinsically add idle time in the transmission of the competing frames. If we neglect the switching latency (time required by the switch fabric to forward a frame when there is no contention due to other frames), the delay imposed on the transmission of the first arriving competing frame f_i^1 is equal to transmission time of this frame, i.e. $\frac{s(f_i^1)}{BW}$. Recall that the start of transmission of the first egress frame defines α^* (see Figure III.6). Considering $\omega^* = 0$, the larger the first egress frame, the larger the initial idle time, and consequently, the larger α^*. Bauer et al. defines the concept of "advance" in [BSF12a] and proves that there exists no scenario in which the sum of idle times on the output link is larger than the length of the largest frame (assuming no idle time on the input links). Therefore, the largest idle time occurs when the first ingress (and consequently first egress) frame is the largest. The worst case scenario presented in Figure III.6 depicts such idle time from ω^* until α^*. We define f^{max} as the largest competing frame and present Equation III.11 to compute the largest idle time in the worst case scenario:

$$idle_v^* = \alpha^* = t(s(f^{max})) \tag{III.11}$$

where:

$$f^{max} = f_k \mid s(f_k) = \max_{\forall f_k \in \mathcal{CF}^{v,o}} s(f_k)$$

III.2.6.1.1.2 Worst case interference The largest possible interference occurs when all HP-frames delay the egress of P-frames stored in the P-buffer. Therefore, in the worst case scenario, all HP-frames ingress from their respective input links and egress before the first P-frame starts transmission. Equation III.12 computes the worst case interference.

$$interf_v^* = t(\sigma_{ALL}^{HP}) \tag{III.12}$$

where:

$$\sigma_{ALL}^{HP} = \sum_{\substack{\forall f_k \in \mathcal{CF}^{\bar{v},o} \\ \forall \bar{v} \in \mathcal{V}^{o,HP}}} s(f_k)$$

For the analysis of virtual links with the highest priority among all competing frames, no HP-frame exists and consequently $interf_v^* = 0$.

III.2.6.1.1.3 Worst case blocking Blocking may occur in three different combinations:

1. P-frame blocks a HP-frame
2. L-frame blocks a HP-frame
3. L-frame blocks a P-frame

For a P-frame to block a HP-frame, this P-frame must egress before β (we omit the scenario index in order to not restrict the analysis to any type of scenario). According to Properties ii) and iii) of Theorem III.1, no P-frame egresses before β^* (in the worst case scenario). Therefore, we do not account for blocking caused by P-frames in the worst case scenario.

Blocking caused by an LP-frame implies the transmission of this frame. This transmission occurs only when all the following conditions hold:

- no HP-frame or P-frame is completely stored in the respective buffer, and
- at least one LP-frame is completely stored in the respective buffer, and
- no frame occupies the output link, i.e. the output link is idle.

The largest blocking occurs when the largest LP-frame starts transmission a (negligibly) small amount of time (ϵ) before a HP-frame or a P-frame is completely stored in the respective buffers. The largest blocking time is equal to the time required to transmit the largest LP-frame, as presented in Equation (III.13). We assume that in the worst case scenario, the largest LP-frame blocks the first arriving HP-frame, which in turn postpones the start of the transmission of the interfering frames (LP-frames).

$$block_v^* = t(s(f^{LP,max})) \tag{III.13}$$

For the analysis of virtual links with the lowest priority among all competing frames, no LP-frame exists and consequently $block_v^* = 0$.

III.2.6.1.2 Computation of θ^{*P} In order to compute the smallest value for θ^{*P}, we define θ_i^{*P} as the arrival time of the last P-frame of each input link i. Consequently, the arrival time of the last P-frame among all input links is:

$$\theta^{*P} = \max_{\forall i \in \mathcal{I}^o} \theta_i^{*P} \tag{III.14}$$

We use the term *sequence* (seq_i) to define the set of competing frames that ingress from the same input link i, and *gap* (gap_i) to define the idle time on input link i from ω^* until the start of arrival of the first frame on that input link. Equation (III.15) computes the size of a sequence:

$$s(seq_i) = \sum_{\substack{\forall f_k \in \mathcal{CF}^{\bar{v},o} \\ \forall \bar{v} \in \mathcal{V}^{o,\bar{P},i} \\ \forall \bar{P} \in \mathcal{P}^o}} s(f_k) \tag{III.15}$$

For a scenario with no idle time on the input links, Equation (III.16) computes the arrival time of the last frame on an input link (θ_i^{*P}) by adding the time required to receive the sequence on this input link to the time when the first frame of this input link starts transmission, as presented in Equation III.16:

$$\theta_i^{*P} = gap_i + t(s(seq_i)) \tag{III.16}$$

According to Equation (III.16), the only term on the computation of θ_i^{*P} that depends on the arrival order of frames is gap_i, and consequently the smallest value of gap_i leads to the smallest value of θ_i^{*P}. We recall that, in order to ensure the largest idle time, the largest frame egresses first, which per definition ingresses at α^*. We assume that, if more than one frame ingresses at the same time, the tie breaking rule always favors the largest backlog accrual (see Section III.2.8). The value of gap_i is equal to:

$$gap_i = \alpha^* - t(s(f_i^{max})) \tag{III.17}$$

According to Equation (III.16) and (III.17), the larger $s(f_i^1)$ the smaller θ_i^{*P}. Therefore, in the worst case scenario, the largest frame of each input link is the first frame to ingress on this input link.

III.2.6.2 Mutually Exclusive Characteristics

For a some combinations of the competing frames, some characteristics used to describe the worst case scenario in the previous sections become mutually exclusive, i.e. they do not occur in the same scenario. Next, we list these mutual exclusive characteristics and show that they do not result in any optimistic computation of the P-buffer backlog upper bound. In order to analyze the impact of these mutual exclusive characteristics on the buffer backlog, we use "fabricated scenarios", i.e., scenarios in which the egress of frames does not strictly respect the dispatching policy of the output port, e.g., assuming that a frame might be transmitted before its arrival or frames do not egress despite their complete arrival. Notice that no additional computation needs to be carried out to account for the impact of those mutually exclusive conditions on the P-buffer backlog upper bound.

III.2.6.2.1 The first arriving frame is the largest among all frames vs. the first arriving frame is a LP-frame In Section III.2.6.1.1.1 we show that, in order for the idle time to be the largest, *the largest frame among all competing frames* must be the first arriving frame. Similarly, Section III.2.6.1.1.3 shows that for the largest blocking time to occur, *the largest LP-frame* must be the first arriving frame. However, there may exist sets of competing frames, in which *the largest frame among all competing frames* and *the largest LP-frame* are not the same frame. For such sets of competing frames, if the largest LP-frame and the largest frame among all competing frames do not ingress from the same input link, our method leads to no pessimism or optimism. In this case, our method assumes that the LP-frame ingresses a negligible amount of time (ϵ) earlier and is thus transmitted first. Figure III.8 depicts such a scenario. Conversely, if these two frames ingress from the same input link, we assume a fabricated scenario in which the largest frame is the first ingress frame and immediately after its complete arrival, the largest LP-frame starts transmission. Figure III.9 depicts such a scenario. In this fabricated scenario, both the largest idle and blocking time occur. This scenario imposes some pessimism on the P-buffer backlog upper bound computation. This pessimism is less than or equal to the difference between the size of the largest competing frame and the largest LP-frame.

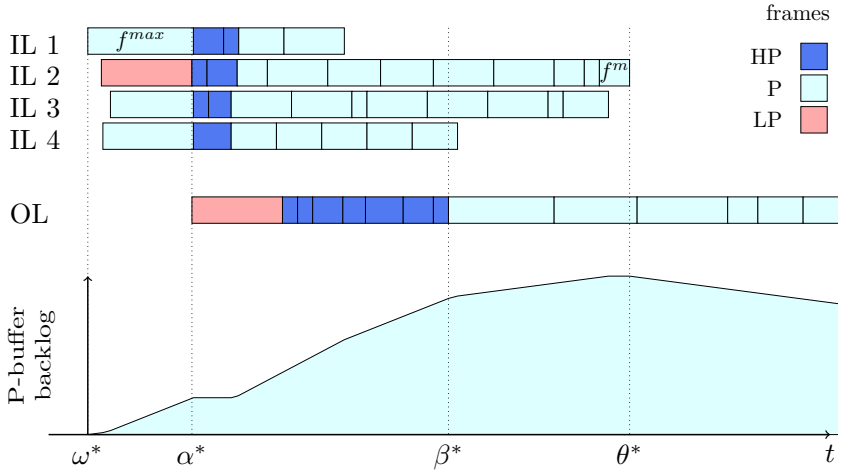


Figure III.8: *Largest P-frame and largest frame among all competing frames egress from different input links*

III.2.6.2.2 No idle time occurs on the input links vs. idle time occurs before the arrival of the last arriving frame For the computation of θ_i^{*P}, we assume in Section III.2.6.1.2 that *no idle time occurs on the input links* during the arrival of competing frames. Property v) of Theorem III.1, however, states that a frame of the virtual link under analysis must be the last ingressing frame. Enforcing the latter property might require assuming *some idle*

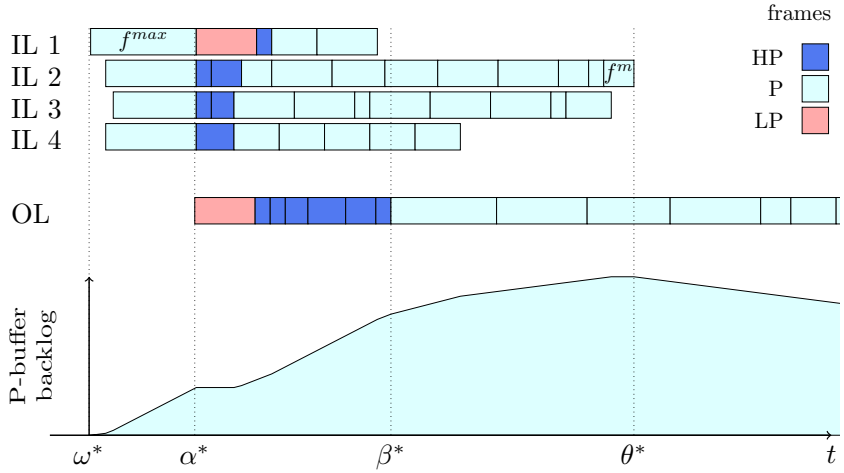


Figure III.9: *Fabricated scenario in which the largest P-frame and largest frame among all competing frames egress from the same input link*

time before the arrival of f^m. This mutually exclusive condition occurs if the sequence in which the virtual link under analysis ingress is not the one leading to the maximum θ_i^{*P}. Figure III.10 depicts a worst case scenario with idle time in the input link IL 4. Assuming idle time in the input link to enforce f^m to be the last ingressing competing frames does not lead to any optimism or pessimism if, by "adding" this idle time, θ^{*P} does not change.

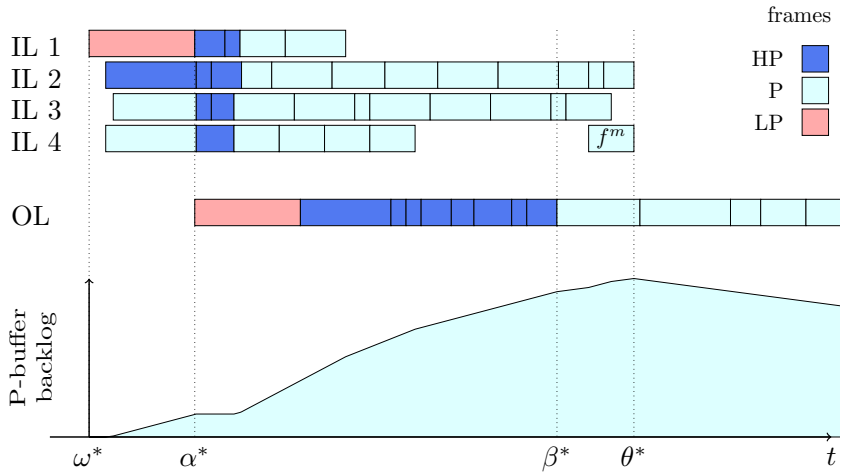


Figure III.10: *Idle time on the input link enforcing* f^m *to be the last arriving competing frame does not impact the computation of the P-buffer backlog upper bound computation*

III.2.6.2.3 A frame of the virtual link under analysis is the last arriving frame vs. a frame of the virtual link under analysis is the first arriving frame For the same reasons presented in Section III.2.6.2.2, a *frame of the virtual link under analysis must be the last ingressing*

frame. However, as presented in Section III.2.6.2.1, in order for the idle time to be the largest, *the largest frame among all competing frames* must be the first arriving frame. Therefore, if the set of competing frames is such that there exists only one frame of the virtual link under analysis and this is the largest among all frames, we assume that this frame is the last to ingress and that the values of θ_i^{*P} (for each input link) and α^* do not change. In this scenario, f^m is the only competing frame of the virtual link under analysis and is the largest frame among all competing frames. Notice that, to enforce α^* and θ_i^{*P}, this fabricated scenario postpones the egress of the three first frames ingressing from input link IL 2. This assumption adds some pessimism to the computation of the upper bound for the backlog of P-buffer. This pessimism is less than or equal to the difference between the size of the frame of the virtual link under analysis and the size of the second largest frame.

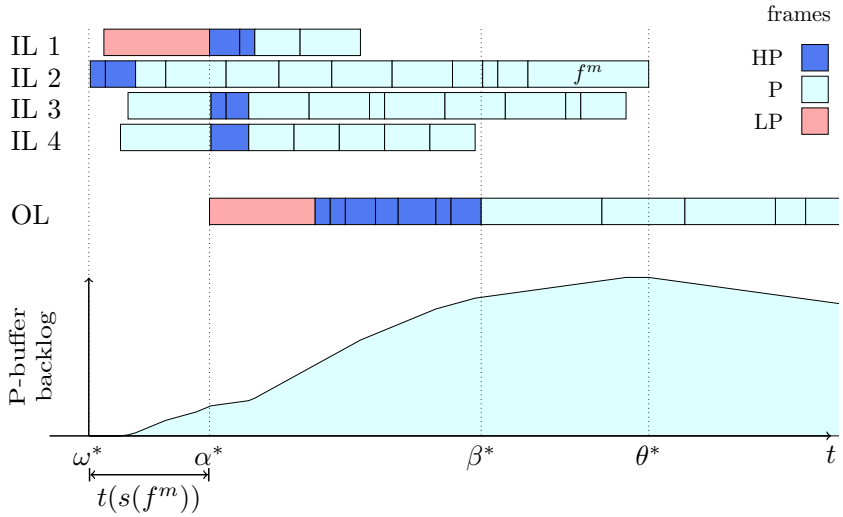


Figure III.11: *Fabricated scenario in which α^* and θ_i^{*P} do not change for a set of competing frames as described in Section III.2.6.2.3*

III.2.7 Upper Bound Computation

The next two sections present the last steps in the computation of an upper bound for the backlog of a buffer in an AFDX network: Section III.2.7.1 presents the upper bound computation for the buffer backlog encountered by a frame of *one virtual link*, and Section III.2.7.2 presents the computation of an *upper bound for the backlog of the buffer under analysis*, taking into account all virtual links that egress an output port through this buffer.

III.2.7.1 Buffer Backlog Upper Bound encountered by one Virtual Link

Recall that $\Delta^{*P} = d(\theta^{*P} - \beta^*)$. The computation of the P-buffer backlog upper bound is split into two cases. If $\Delta^{*P} \leq 0$, then there exists a scenario in which, at a given point

in time, all competing P-frames are stored in the buffer (Figure III.12).

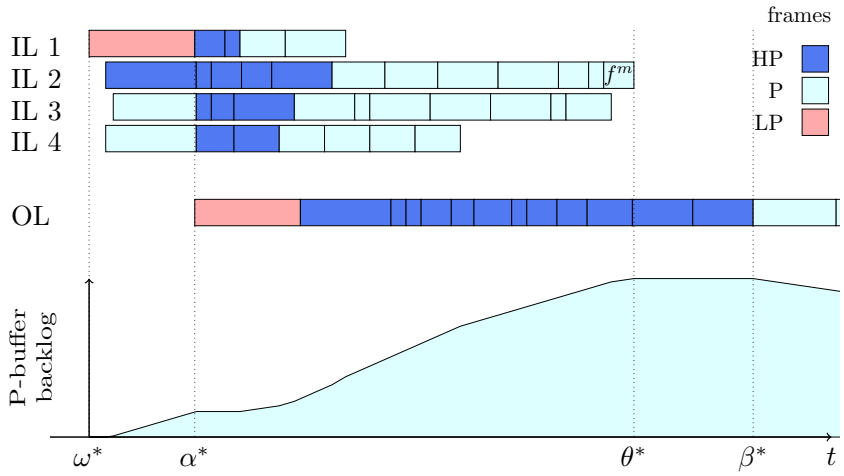


Figure III.12: *Worst case scenario in which* $\Delta^{*P} < 0$

Consequently, the backlog upper bound is equal to the sum of the sizes of all competing P-frames. If $\Delta^{*P} > 0$, then Δ^{*P} represents the amount of data transmitted during the single type 2 interval that occurs until θ^{*P} (Figure III.6). Therefore, the backlog upper bound is equal to sum of the sizes of all competing P-frames minus Δ^{*P}.

Considering $(\Delta^{*P})^+ = \max(0, \Delta^{*P})$, we compute the upper bound for the P-buffer backlog encountered by a frame of the virtual link under analysis (line 5 of Algorithm 1) as:

$$B^{o,P,v,max} = \sigma^P_{ALL} - (\Delta^{*P})^+ \tag{III.18}$$

III.2.7.2 Backlog Upper Bound for the Buffer Under Analysis

The computations presented in Equations (III.10) - (III.18) provide an upper bound for the backlog encountered by a frame of a virtual link with the same priority as the buffer under analysis (line 5 of Algorithm 1). After considering the upper bounds encountered by each egress virtual link with priority P (*for each* loop between lines 3 and 5 of Algorithm 1), Equation (III.19) computes the upper bound for the P-buffer backlog on a switch output port (line 6 of Algorithm 1):

$$B^{o,P,max} = \max_{\forall v \in \mathcal{V}^{o,P}} (B^{o,P,v,max}) \tag{III.19}$$

III.2.7.3 Summary of Upper Bound Computation

Algorithm 2 P-buffer backlog upper bound encountered by a frame of virtual link v

function ComputeBacklog($\mathcal{N}, \mathcal{CF}^{v,o}, P$)
  $\beta^* \leftarrow$ ComputeBetaWorstCase($\mathcal{N}, \mathcal{CF}^{v,o}, P$)
  $\theta^{*P} \leftarrow$ ComputeThetaWorstCase($\mathcal{N}, \mathcal{CF}^{v,o}, P$)
  $\Delta^{*P} \leftarrow d(\theta^{*P} - \beta^*)$
  $\sigma^P_{ALL} \leftarrow$ ComputeSumOfSizesOfCompetingP($\mathcal{N}, \mathcal{CF}^{v,o}, P$)
  return $\sigma^P_{ALL} - (\Delta^{*P})^+$

function ComputeBetaWorstCase($\mathcal{N}, \mathcal{CF}^{v,o}, P$)
  $f^{max} = f_k \mid s(f_k) = \max_{\forall f_k \in \mathcal{CF}^{v,o}} s(f_k)$
  $idle^*_v \leftarrow t(s(f^{max}))$
  $\sigma^{HP}_{ALL} \leftarrow \sum_{\substack{\forall f_k \in \mathcal{CF}^{\bar{v},o} \\ \forall \bar{v} \in \mathcal{V}^{o,HP}}} s(f_k)$
  $interf^*_v \leftarrow t(\sigma^{HP}_{ALL})$
  $s(f^{LP,max}) = \max_{\substack{\forall f_k \in \mathcal{CF}^{\bar{v},o} \\ \forall \bar{v} \in \mathcal{V}^{o,LP,\bar{i}} \\ \forall \bar{i} \in \mathcal{I}^o}} s(f_k)$
  $block^*_v \leftarrow t(s(f^{LP,max}))$
  return $idle^*_v + interf^*_v + block^*_v$

function ComputeThetaWorstCase($\mathcal{N}, \mathcal{CF}^{v,o}, P$)
  $s(seq_i) \leftarrow \sum_{\substack{\forall f_k \in \mathcal{CF}^{\bar{v},o} \\ \forall \bar{v} \in \mathcal{V}^{o,\bar{P},i} \\ \forall \bar{P} \in \mathcal{P}^o}} s(f_k)$
  $f^{max}_i = f_k \mid s(f_k) = \max_{\substack{\forall f_k \in \mathcal{CF}^{\bar{v},o} \\ \forall \bar{v} \in \mathcal{V}^{o,\bar{P},i} \\ \forall \bar{P} \in \mathcal{P}^o}} s(f_k)$
  $gap_i \leftarrow \alpha^* - t(s(f^{max}_i))$
  $\theta^{*P}_i \leftarrow gap_i + t(s(seq_i))$
  return $\max_{\forall i \in \mathcal{I}^o} \theta^{*P}_i$

function ComputeSumOfSizesOfCompetingP($\mathcal{N}, \mathcal{CF}^{v,o}, P$)
  return $\sum_{\substack{\forall f_k \in \mathcal{CF}^{\bar{v},o} \\ \forall \bar{v} \in \mathcal{V}^{o,P,\bar{i}} \\ \forall \bar{i} \in \mathcal{I}^o}} s(f_k)$

III.2.8 Discussion

The analysis presented in Sections III.2.6 and III.2.7 assumes a switch implementation in which data transfers from and to output buffers occur bit-wise, i.e., frames are copied to output buffers as they ingress and removed from these buffers as they egress, bit by bit. We name this implementation *Design 1*. Next we consider two other possible implementation designs (*Design 2* and *Design 3*) and show how the backlog upper bound in these implementations relates with the results presented in Section III.2.7.

We assume, for *Design 2*, that ingress frames are first completely stored into an input buffer and then copied "at once" to the respective output buffer. The output buffer memory allocated to a frame is only freed after the complete transmission of this frame.

For *Design 3*, we assume that the amount of data required to store the complete frame is reserved on the output buffer as soon as the first bit of the frame ingresses the switch. Similar to *Design 2*, we assume that the memory allocated to a frame is only freed after the complete transmission of this frame. Figure III.13 presents a graphic representation of the buffer backlog behaviour for each implementation design: the upper part depicts the ingress and egress of a P-frame and the lower part shows the respective P-buffer backlog for each of the three previously described implementation designs.

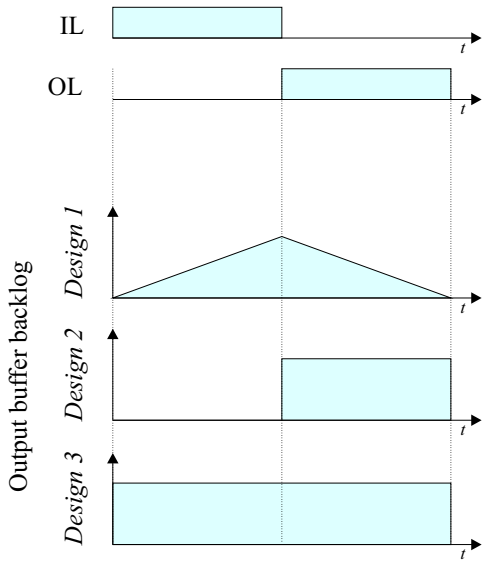


Figure III.13: *P-buffer backlog behaviour of each implementation design during ingress and egress of a P-frame*

The buffer backlog accrual due to the ingress of frames, for *Design 2* switches, is less than or equal to the one presented in our paper. The optimism w.r.t. our method is less than or equal to the sum of the largest competing frame of each input link. During the egress of frames, *Design 2* switches lead to more pessimistic backlog accrual than the one presented in our paper. This pessimism is less than or equal to the size of the largest competing frame. Considering that only one frame egresses at a time, we can

compute an upper bound for the buffer backlog of *Design 2* switches by adding the size of the largest competing frame to the result from Equation (III.19).

The buffer backlog accrual for switches implemented according to *Design 3* is, in comparison to *Design 1*, larger both during ingress and egress of frames. During the ingress of a frame, the pessimism of *Design 3* is less than or equal to the size of this frame. Considering that frames ingress from multiple input links, the total pessimism due to ingress of frames is less than the sum of the largest competing frame of each input link. Similarly, during the egress of any frame, the pessimism is less than or equal to the size of this frame. Thus, we can compute an upper bound for the buffer backlog of switches implemented according to *Design 3* by adding the result from Equation (III.19) to the sum of the size of the largest competing frame of each input link plus the size of the largest competing frame among all frames.

III.3 Experiments

We implemented a tool to use our proposed method to compute the upper bounds for the buffer backlog of AFDX networks with multiple priority traffic. This tool starts by using the trajectory approach to compute the required competing frames. Then, it computes an upper bound for the backlog of each buffer on the switch output port, as presented in Section III.2.

In our experiment, we are interested in measuring the time required to compute the buffer backlog upper bound applying our proposed method. We assume that the network designer analyzes the end-to-end latency on the network using the trajectory approach, or any similar method that computes the busy period of frames along their path. As part of the end-to-end latency analysis, the trajectory approach computes most of the values required by our method. Therefore, the time added for the computation of the buffer backlog upper bound is minimum.

For our experiment, we created a set with 500 virtual links and 3452 paths. We uniformly assign one out of three priority levels to the virtual links. The network topology in this experiment resembles the topology of the Avionics Network Laboratory at the chair of real-time systems in the University of Kaiserslautern, which in turn is similar to the publicly available topology of the Airbus A380 [But10]. Considering that the same traffic is present in the redundant AFDX networks, we do not consider the redundant network in our analysis. The route of all virtual links obey the path restriction assumed by the trajectory approach (see Section II.3.2.1.3).

In the aforementioned experiment, it takes our tool 60 minutes to compute the end-to-end latency of 49 virtual links that egress the output port under analysis, and 20 seconds to compute the upper bounds for the backlog of the three buffers (one per priority level) on that output port. We observe from this experiment that the computational cost required by our method is negligible.

III.4 Summary

In this chapter, we presented a method to compute an upper bound for the backlog of every buffer on an AFDX network. We showed that contention occurs on a switch output port every time multiple frames try to egress this port at the same time. The contention issue is addressed by means of buffering in AFDX networks. Considering that safety-critical applications of avionics systems exchange data via AFDX network, overflow of these buffers must be avoided at all cost. Even though the AFDX standard recommends a minimum amount of memory to be reserved for the output buffers, this amount does not represent a safe bound for every network configuration. Further, this recommendation does not specify how the reservation should be shared among buffers of different priorities.

The current AFDX standard allows for the classification of network traffic in two priorities: *high* and *low*. However, some existing commercial solutions extended the number of used priorities allowing, for instance, frame scheduling according to the rate monotonic policy, if the number of existing priorities is larger than or equal to the number of used BAGs. In the approach presented in this chapter, we assumed an AFDX network in which virtual links are classified with any number of priority levels. We provided the computation of an upper bound for each buffer (one per priority level) in the network.

We investigated, for each output buffer, the largest backlog encountered by probe frames (called f^m) of each virtual link with the same priority as that buffer. We analyzed the largest busy period encountered by the probe frame of a virtual link and presented the arrival properties of the frames in the busy period (called competing frames) that lead to the largest backlog faced by the probe frame. In order to facilitate the investigation of those properties, we introduced the concept of *types of intervals*. Each type of interval has well defined properties w.r.t. ingress and egress of frames.

We showed that, according to the defined properties, only four types of intervals exist. We used the four types of interval to describe the properties of the scenario (ingress and egress order of frames in the largest busy period) leading to the largest backlog encountered by a frame of a virtual link.

We formalized the worst case scenario description in Theorem III.1 and proved that there exists no other scenario in which the probe frame encounters a buffer backlog larger than the one in the worst case scenario. Once the worst case scenario is identified, we showed how to compute an upper bound for the buffer backlog encountered by a frame of one and then of all virtual links that egresses a switch through the output port in which the buffer under analysis belongs to. This computation provided an upper bound for the backlog of the buffer under analysis.

Our approach assumed that data transfers from and to output buffers occur bit-wise inside the switches, i.e., frames are copied to output buffers as they ingress and removed from these buffers as they egress bit by bit. Output buffers of switches that do not follow this assumption may encounter different backlog upper bounds. Hence, we also discussed in this chapter methods on how to apply the results achieved under our assumptions onto switches in which internal data transfer occurs differently.

The results from our simulations showed that, the computation time required exclusively by our method is negligible when compared to the time required to compute the end-to-end delay in the same network by an state-of-the-art approach.

Chapter IV

TTEthernet Background and State-of-the-Art in Time-Triggered Schedulers

This chapter describes the fundamental properties of Time-Triggered Ethernet (TTEthernet) and the related work on the computation of schedules for time-triggered systems.

Section IV.2 presents the terminology used to address data transmission over TTEthernet. In Section IV.3, we present the properties of TTEthernet required in Chapter V. Then, in Section IV.4, we present the state-of-the-art methods used to generate schedules for time-triggered systems. Section V.6 concludes this chapter with a summary.

IV.1 Introduction

Time-triggered Ethernet [tim16] and [Kop08], is an extension of the switched Ethernet (IEEE 802.3) that allows for deterministic data communication and standard Ethernet traffic to coexist in the same network. According to the temporal guarantees, TTEthernet defines three classes of messages: Best Effort (BE), Time-Triggered (TT), and Rate-Constrained (RC).

Best Effort BE-messages are the "normal" Ethernet messages, and therefore have no temporal guarantees.

Rate-Constrained RC-messages follow the AFDX standard, ensuring a minimum preconfigured bandwidth for each RC-message. Chapter II presents details about AFDX.

Time-Triggered TT-messages are transmitted strictly periodically, thus providing deterministic communication channels. The dispatching times of the frames of every TT-message are defined off-line in a conflict-free schedule. In order to ensure the correct dispatch time of TT-frames in every network node, all nodes involved in the time-triggered communication must agree on a common time base. For this reason, TTEthernet implements a transparent clock synchronization protocol. Section IV.3.1 describes the properties of TTEthernet clock synchronization protocol relevant to the schedule of time-triggered messages.

IV.2 Terminology

In this dissertation, we use three terms to address the data transmission over TTEthernet at different levels: *frame*, *virtual link*, and *message*. Notice that the following terminology is slightly different from the one previously described in Section IV.1. Figure IV.1 depicts the relationship between these three terms.

Frame: We adhere to the OSI model and consider *frame* the Protocol Data Unit (PDU) at the data link layer.

Virtual link: A time-triggered *virtual link* is a virtual communication channel between one source and one or multiple destination nodes providing not only bandwidth reservation but also deterministic transmission for its frames. Frames of a TT virtual link are transmitted strictly periodically.

Message: *Message* is the actual information generated at application level, e.g., a sensor or an actuator value, the result of a computation or an encoded video frame. A large message may span multiple frames (see the example in Figure IV.1). Similarly, multiple messages might be merged into the single frame. In the simplest case, one frame carries one message. The analysis of methods to group or segment messages into frames is out of the scope of this dissertation.

Phase: We use the term *phase*, represented by ϕ, to refer to the dispatch time of the first frame of a virtual link w.r.t. the start of a TTEthernet schedule. The phase assigned to a virtual link defines the dispatching time of each frame of this virtual link: due to the strictly periodic transmission of frames, the dispatching time of the k-th frame of a virtual link v scheduled at phase ϕ_v occurs at a time $t_v^k = \phi_v + (k-1) \times T_v$, where T_v is the virtual link period. Figure IV.1 shows the phase and the scheduled transmission windows of each frame of a virtual link.

Physical link: *Physical link* is the term used to refer to a unidirectional connection between two nodes in the network. Hence, the full-duplex (bi-directional) connection between two nodes is modeled as two (unidirectional) physical links.

Schedule: We call *schedule*, the phases assigned to TT virtual links. Notice that, whereas a scheduling table contains the transmission time of each TT-frame, a schedule contains the phase of the virtual link.

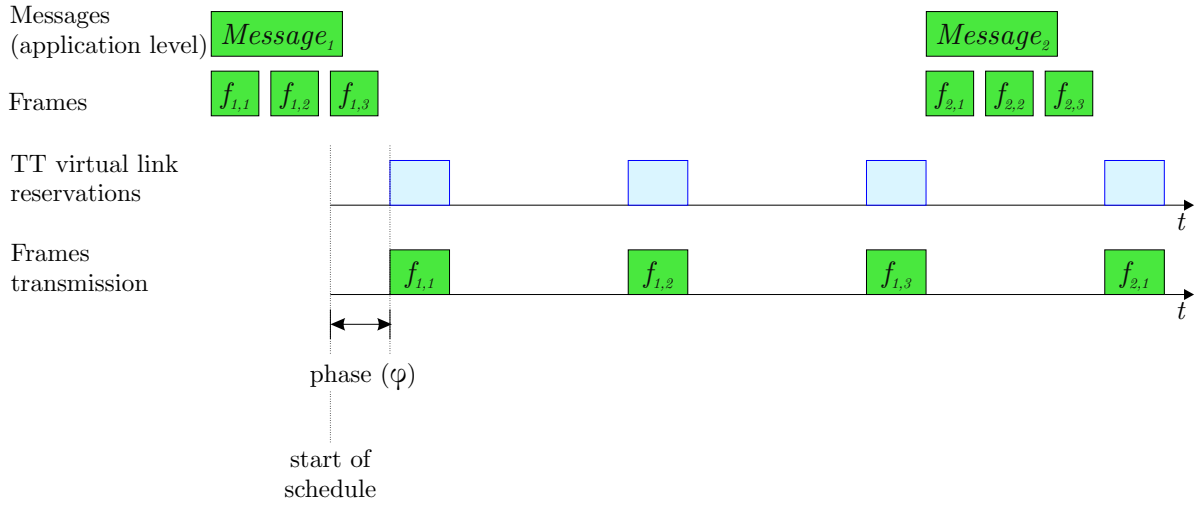


Figure IV.1: *Relationship between message, frame, virtual link and phase.*

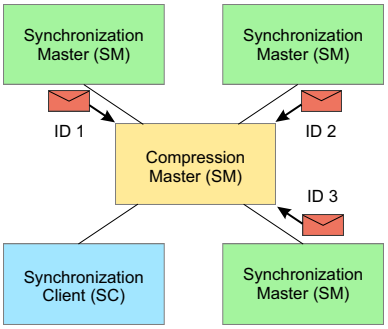
IV.3 Relevant Properties of Time-Triggered Ethernet

IV.3.1 Clock Synchronization

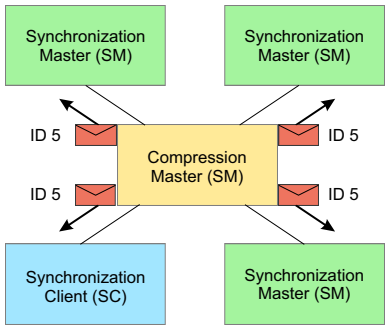
TTEthernet allows the nodes connected to the network to have a common notion of time by providing a transparent, fault-tolerant clock synchronization protocol. This protocol is classified as *transparent* because, despite using the same medium (the network) to exchange the clock synchronization messages, other types of traffic coexist in the same network, e.g., rate-constrained, best effort.

The nodes involved in the clock synchronization protocol are divided into three categories: synchronization masters (SM), synchronization clients (SC), and compression

masters (CM). These nodes exchange a special type of messages, called Protocol Control Frames (PCF) to establish the clock synchronization. We divide the analysis of the clock synchronization protocol in two steps, depicted in Figure IV.2. In a first steps, each SM sends a PCF to the CM nodes (messages ID 1, ID 2, and ID 3 in Figure IV.2a). Then, the CM nodes calculate an average value from relative arrival times of these PCFs and send a PCF to all nodes involved in the clock synchronization protocol (messages ID 1, ID 2, and ID 3 in Figure IV.2b), i.e., SCs and SMs. This message exchange occurs periodically, at time intervals called *integration cycles*. A set of integration cycles defines the *cluster cycle*, which is the hyper period of the time-triggered schedule. Figure IV.3 presents the relationship between, cluster cycle, integration cycle, and the points in time when the exchange of PCFs occurs. Notice that the exchange of messages used by the clock synchronization protocol occurs in a time interval limited by the start of an integration cycle and $t_{syncEnd}$.



(a) *Synchronization masters send PCFs to the compression master.*



(b) *Compression master sends PCFs to SM and SC.*

Figure IV.2: *Exchange of messages for the TTEthernet clock synchronization protocol (adapted from [tim16]).*

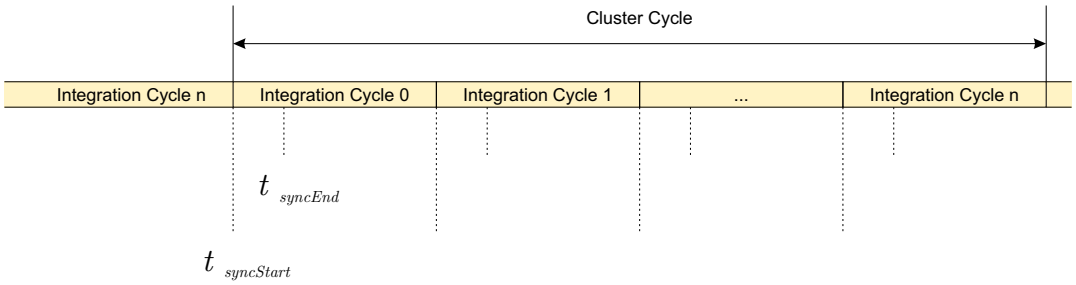


Figure IV.3: *Relationship between cluster cycle, integration cycle, and transmission of PCFs.(adapted from [tim16])*

For more details on the clock synchronization protocol used in the TTEthernet, we refer the interested reader to the respective AES standard [tim16].

IV.3.2 Scheduling TTEthernet Transmission Windows

Scheduling a TT virtual link means reserving a time window (transmission windows) at a scheduled point in time for the transmission of each frame of that virtual link on each physical link in its path. The problem of scheduling TT-frames in a physical link is similar to scheduling *non-preemptive strictly periodic* jobs in *offset*[1] *free systems* [Goo03] or *independent operations strictly periodically* [Kor92]. Next, we address the common concepts of these two domains: job and frame scheduling.

Offset free systems Goossens describes periodic *offset free systems* in [Goo03] as systems in which jobs have no definite requirement about starting times. In such systems, an off-line scheduler assigns these starting times. Earliest starting times of jobs in such system are separated by task period. Similarly, definite transmission windows start times in TTEthernet networks are not requirements given by applications, they are rather computed by the TTEthernet off-line scheduler.

Strictly periodic The separation between the start of execution of any two consecutive jobs of a task in *strictly periodic* systems is equal to the task period. Similarly, the reserved transmission windows of any two consecutive TT-frames of the same virtual link is separated by the virtual link period.

Non-preemptive In this chapter, we assume that the transmission of a TT-frame is not interrupted until the frame is completely transmitted. More specifically, we assume that the frame integration strategy used by TTEthernet is *timely block* (see Section IV.3.4). This strategy is similar to non-preemptive job scheduling.

A valid TT schedule for a given physical link shall not contain overlapping transmission windows. This property ensures no contention among frames of TT virtual links. Nevertheless, contention might occur due to the transmission of ET-frames, since their transmission times are not known in advance. Next section presents how TTEthernet addresses this contention.

IV.3.3 Contention

TTEthernet defines in [tim16] and [Kop08] three strategies to address contention and consequently, integrate the transmission of TT- and ET-frames:

Timely block This strategy ensures that, inside a transmission window, only the scheduled TT-frame can be transmitted. Consequently, an ET-frame does not start transmission if its (partial) transmission would occur inside a scheduled TT transmission window. The transmission of such an ET-frame is deferred to a point in time in which this frame can be completely transmitted outside the scheduled TT transmission windows (see Figure IV.4). On the one hand, this strategy leads to underutilization of network bandwidth: TT-frames might not completely occupy the reserved transmission window, or some idle time might be enforced to

[1]The term *offset* used in [Goo03] is called *phase* in this dissertation.

avoid an ET-frame to be transmitted inside a transmission window. On the other hand, this strategy allows TT-frames to be transmitted exactly at the scheduled point in time ensuring low jitter for TT-frames.

Preemption This strategy allows for the transmission of ET-frames inside scheduled transmission windows. If the scheduled TT-frame becomes ready for transmission while another frame occupies the reserved transmission window, the transmission of the ET-frame is preempted and the TT-frame starts transmission until completion. After the complete transmission of the TT-frame, the transmission of the preempted ET-frame restarts. This strategy allows for high network bandwidth utilization, since the reserved transmission window might be used by ET-frames. An important issue to be addressed is, the need of a mechanism to allow the receivers to identify the preempted frame as a truncated one, and in case of resumed preemption, to reconstruct the preempted frame.

Shuffling Similar to preemption, this strategy allows for the transmission of ET-frames inside TT scheduled transmission windows. However, if the scheduled TT-frame becomes ready for transmission while an ET-frame is transmitted, the latter is not preempted and the scheduled TT-frame is delayed until the ET-frame completes transmission. Under this strategy, in the worst case, a TT-frame is blocked by the largest ET frame. On the one hand, the shuffling strategy allows for high bandwidth utilization, since ET-frames can be transmitted whenever the network is idle and is not preempted (and consequently, need not be retransmitted). On the other hand, the major drawback of this strategy is the large jitter that might be imposed on TT-frames.

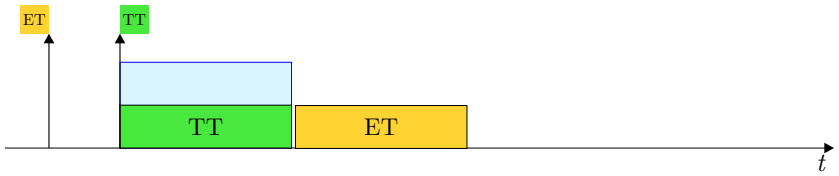
IV.3.4 Transmission Window Implementations

Heilmann and Fohler describe in [HF17] two possible implementations for a valid transmission of a TT-frame inside a scheduled window.

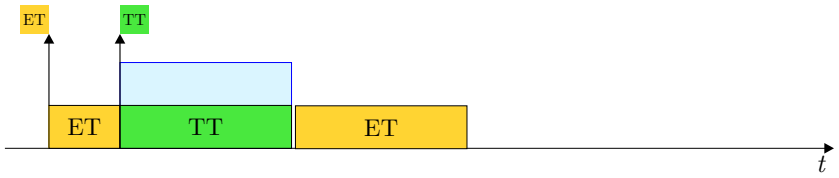
Last Bit Strict (LBS) This implementation requires the last bit of a TT-frame to be transmitted inside the scheduled window. In this chapter, we assume a more restrictive version of LBS in which TT-frames start and finish their transmission inside the scheduled window.

First Bit Strict (FBS) This implementation requires the first bit of the scheduled TT-frame to be transmitted inside the scheduled window. The point in time when the scheduled TT-frame completes transmission is not relevant. Nevertheless, the completion occurs not later than the point in time when the scheduled window ends plus the time required to transmit the TT-frame.

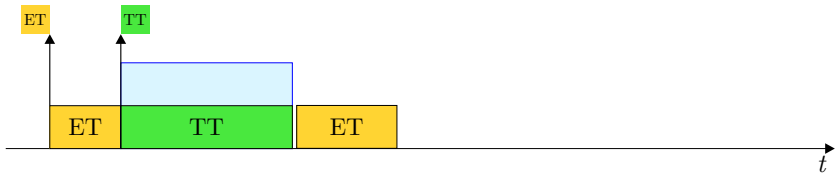
Figure IV.4 depicts the relationships between integration strategies and transmission window implementations. In this figure, arrows pointing up represent the moment when an ET- or TT-frame is ready for transmission, and the light blue rectangles represent the scheduled transmission windows for TT-frames.



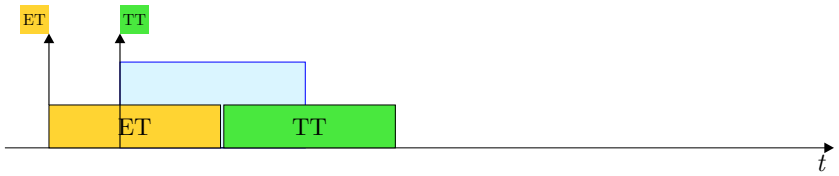
(a) *Last bit strict and timely block*



(b) *Last bit strict and preemption*



(c) *Last bit strict and resumed preemption*



(d) *First bit strict and shuffling*

Figure IV.4: *Impact of transmission window implementations on frame integration strategies.*

In this chapter, we assume the implementation of *timely block* and the restrictive version of *last bit strict* Therefore, we consider that the transmission window length is computed by:

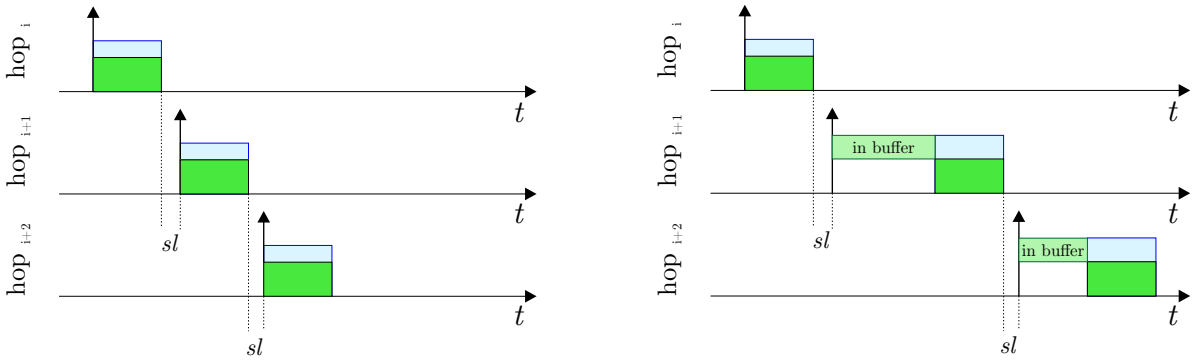
$$tx_{length} = \frac{S_{max} + Preamble + SFD + IFG}{BW} \tag{IV.1}$$

where: S_{max}, representing the largest length of a frame of that virtual link, ranges from 64 to 1518 bytes; and preamble, start frame delimiter (SFD), and inter-frame gap (IFG) have fixed length of 7, 1, and 12 bytes, respectively. We assume in this dissertation that S_{max} in Equation (IV.1) is an input parameter to the scheduler, which might not be the same for each virtual link in the system. The transmission window start time of the first frame of a virtual link (phase) is stored into the TTEthernet scheduling table.

IV.3.5 TT Frames Latency

In order to achieve the *minimum latency* of TT-frames, a TT schedule is constructed such that TT-frames need not to be buffered[2] along their path, i.e., a transmission

[2]Except for the buffering required by store and forward. Due to the store and forward approach used in TTEthernet, switches must store a complete frame before relaying this frame to an output port.



(a) *Minimum latency for TT-frames when transmission windows are scheduled back-to-back along the virtual link path.*

(b) *If TT-frames are not scheduled back-to-back along their path, TT-frames must be buffered, and hence the latency is not minimum.*

Figure IV.5: *Transmission window schedule along a TT virtual link path.*

Table IV.1: *Transformation of sporadic task into periodic reservation.*

Parameter	Sporadic Task	Reservation Periodic Task
Execution Time	C_i	$C_i^{'} = C_i$
Deadline	d_i	$d_1^{'} = C_i$
Minimum Interarrival Time	T_i	$T_i^{'} = \min(l_i + 1, T_i),\ l_i = d_i - C_i$

window starts immediately after the scheduled arrival of a frame. Nevertheless, this is not a mandatory condition.

Heilmann and Fohler show in [HF17] that scheduling TT-frames "back-to-back" might lead to large jitter on RC-frames. Although timing analysis of RC-frames is not part of this dissertation, we describe in Appendix IV.3.6 an approach that converts temporal requirements of RC-frames into TT virtual links thus allowing for RC reservations in the generation of TT schedule.

Figure IV.5 shows two simple examples where a TT-frame is scheduled across three consecutive links: Figure IV.5a depicts a "back-to-back" schedule where no buffering is required and Figure IV.5b shows another valid schedule in which the TT-frame must be buffered and consequently faces a larger latency.

IV.3.6 Converting RC Temporal Requirements into TT Reservations

We describe in this section a method to account for the temporal requirements of RC-frames in the generation of a TT schedule. Mok presents in [Mok83], a method to transform the requirements of sporadic into periodic tasks reservations. Table IV.1 summarizes this method.

Similarly, we convert the requirements of RC virtual links into *reservation* TT virtual links and generate a schedule that accounts for these reservation virtual links. Obviously, the computed phases for these reservation TT virtual links are not part of the schedul-

ing table, they are rather a guarantee that the RC virtual links meet their temporal requirements at run-time.

IV.4 Related Work

The literature provides different approaches that can be applied to compute schedules for TTEthernet networks. In this section we describe the most relevant ones, divided into three types of solutions. Section IV.4.1 presents solutions based on satisfiability modulo theories (SMT) and mixed integer programming (MIP) solvers. Section IV.4.2 presents a method based on the meta-heuristic *tabu search*. And finally, Section IV.4.3 shows the results of strictly periodic scheduling of non-preemptive systems which can be used to schedule TT virtual links. We extend the description of the related work in Section IV.4.3 in order to highlight the principles applied to STSTTN.

IV.4.1 SMT and MIP Solver Approaches

Steiner shows in [Ste10] a method to model the scheduling problem of TT virtual links as a set of SMT constraints. These constraints account not only for the network (set of virtual links, paths, etc.) but also the available memory resources in each traversed switch. The solver provides the schedule for all TT virtual links on the network. However, the main drawback of this approach is the exponentially increasing computational time required for the SMT-solver to synthesize a schedule. Steiner presents the required synthesis time as a function of frames in the system. For the examples presented in [Ste10], depending on the distribution the number of destinations per virtual links, it takes the SMT solver up to 30 minutes to schedule approximately 18.000 frames in a tree topology.

Craciunas and Oliver present in [CO16] an extension to Steiner's work in which they propose a combined schedule of TT tasks and TT virtual links. Further, they propose a MIP formulation which allows for the solver to find a combined schedule while optimizing one(some) parameter(s), e.g., minimizing end-to-end latencies. The results of the tested scenarios show that a SMT solver synthesizes a schedule for a *large*[3] scenario in a tree topology in approximately 1 minute. For a *huge*[4] scenario however, the solver is not able to find a solution after a time-out of 10 hours. Adding the optimization constraint to minimize end-to-end latency seems not to increase the synthesis time significantly.

It is important to notice, however, that the conclusions regarding the required synthesis time presented in both papers should be carefully considered: the presented synthesis time values represent the result of the synthesis of one test set, i.e., another test set with similar properties (period, utilization, etc.) could lead to a different result. Further, the authors do not present details on the network utilization. Nevertheless, the numbers presented in both papers show that these methods do not scale well.

[3]Mesh or ring topology with 8 switches and 48 end systems, or tree topology with 15 switches and 48 end systems

[4]Mesh or ring topology with 16 switches and 192 end systems, or tree topology with 43 switches and 432 end systems

IV.4.2 Tabu Search Meta Heuristics

Selicean et al. present in [TPS15] a method based on the meta-heuristic *tabu search* to schedule TT virtual links. This work also accounts for the temporal constraints of RC virtual links. Their goal is to minimize the end-to-end delay of RC frames by modifying the message packing/ fragmentation into frames, assignment of frames to virtual links, routing of virtual links and the actual (feasible) schedule of TT frames.

The authors present an evaluation section which presents the results achieved after running the proposed method (DOTTS) for 45 minutes in different scenarios with up to 90% network load and up to 37 end-systems. This evaluation shows that, if the tabu search modifies not only the schedule but also the route and packing/fragmentation strategies, DOTTS increases the schedulability of all described message sets. All tested message sets are deemed schedulable with DOTTS. Further, the average end-to-end delay of RC messages notably decreases for some message sets. However, as in the experiments mentioned in Section IV.4.1, each point in the results table represents one message set with the described properties. Moreover, no details, except maximum and minimum values, about the tested periods is mentioned in this paper. Nevertheless, the results present a clear trend for all message sets showing that they are all scheduled within 45 minutes.

IV.4.3 Strictly Periodic Scheduling

This section presents the related work on strictly periodically scheduling, which is used as basis for our scheduler. We divide the scheduling analysis of a TTEthernet network in two steps: first scheduling a pair of tasks, then multiple tasks.

IV.4.3.1 Scheduling a Pair of Tasks

Korst shows in Chapter 4 of his PhD thesis [Kor92] that scheduling non-preemptive strictly periodic tasks in offset free uniprocessor systems (called in his thesis *scheduling independent operations strictly periodically*) is an NP-complete problem in the strong sense. Assuming some restrictions on the message sets however, drastically reduces complexity. For example, Korst shows, also in [Kor92], that, if all tasks periods and execution times are divisible, this scheduling problem can be solved in polynomial time. However, according to the Time-Triggered Ethernet functionality described in the Society of Automotive Engineers standard SAE AS6802 [tim16], no such restrictions are mandates for TT virtual links. Therefore, these restrictions are not assumed by STSTTN.

Goossens shows in [Goo03] that, for any pair of strictly periodic tasks v_i and v_k with periods and worst case execution time T_{v_i}, C_{v_i} and T_{v_k}, C_{v_k} respectively, the largest possible distance between the execution of a job of v_i and the following job of v_k is equal to the greatest common divisor(gcd) of the two periods, i.e., $\gcd(T_{v_i}, T_{v_k})$. We can assume, without loss of generality, that one task is scheduled at time origin, e.g. $\phi_i = 0$. Figure IV.6 depicts an example with possible schedules (including infeasible ones) for two tasks v_1 and v_2 such that $T_{v_1} = 8, C_{v_1} = 2$ and $T_{v_2} = 12, C_{v_2} = 1$. This figure further depicts the concept of equivalent schedules, i.e., schedules in which the execution of jobs

are located at same relative points in time. Goossens summarizes in [Goo03] the concept of equivalent schedules presented in [Kor92] as "schedules that lead to the same periodic behavior". Notice that Figure IV.6 shows that schedules are equivalent whenever ϕ_2 is displaced by $\gcd(8, 12) = 4$ units of time. Korst shows in [Kor92] that, when scheduling two strictly periodic operations(task), all non-equivalent choices for scheduling a task τ_2, considering τ_1 is scheduled, are represented by the set $\{0, 1, 2, \ldots, \gcd(T_1, T_2) - 1\}$. The presented example with two virtual links introduces the intuition behind the use of the greatest common divisor of periods in the TTEthernet scheduling problem.

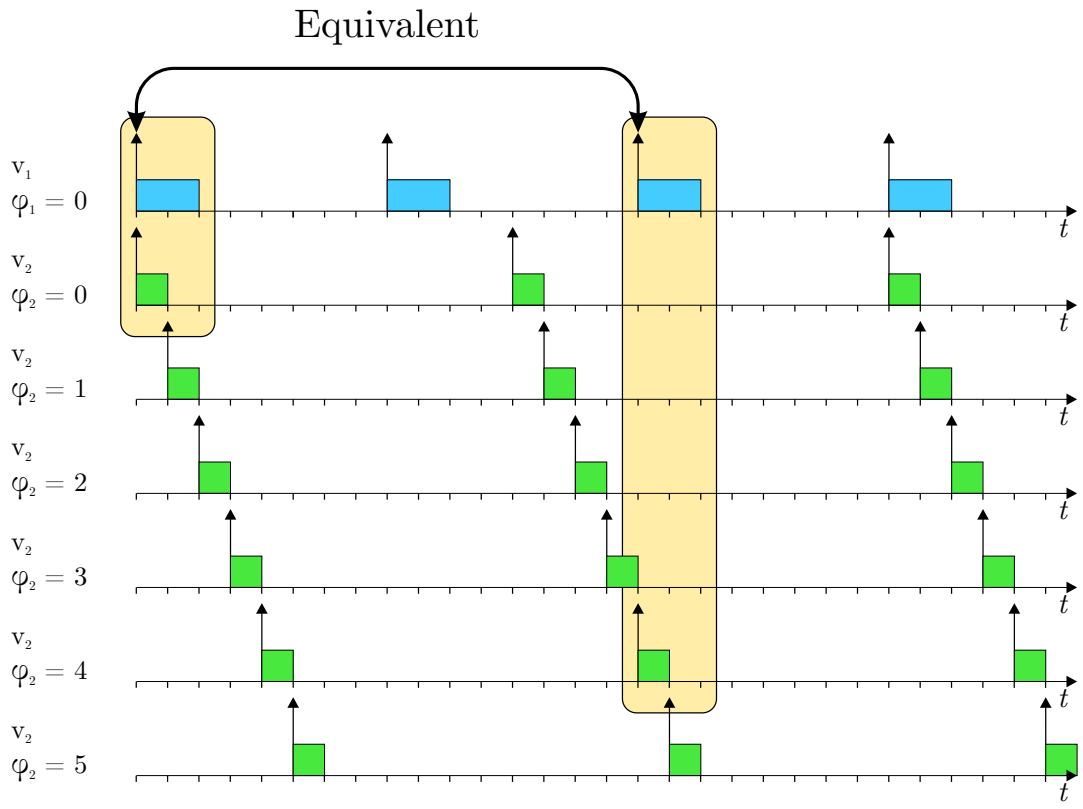


Figure IV.6: *Schedules for two virtual links on the same physical link. v_1 scheduled at $\phi_1 = 0$ and six different schedules for v_2. The number of non-similar schedules is 4.*

For any a pair of strictly periodic tasks, Marouf and Sorel [MS10] present Equation (IV.2) to compute the set of feasible phases (Λ_k) for v_k assuming that v_i is scheduled at ϕ_i.

$$\Lambda_k = \phi_i + n \times \gcd(Tv_i, Tv_k) + m \qquad \text{(IV.2)}$$
$$\forall n \in \mathbb{N}, \forall m \in [C_i, \gcd(Tv_i, Tv_k) - C_k]$$

Equation (IV.3) integrates the principle of non-equivalent schedules presented in [Kor92]

into the computation of feasible phases proposed in [MS10]:

$$\begin{aligned} &\Lambda_k = \phi_i + n \times \gcd(Tv_i, Tv_k) + m \\ &\forall n \in \mathbb{Z}, \forall m \in [C_i, \gcd(Tv_i, Tv_k) - C_k], \text{such that:} \\ &\quad \min\{\Lambda_k\} \geq 0, \max\{\Lambda_k\} < \gcd(Tv_i, Tv_k) \end{aligned} \tag{IV.3}$$

Korst presents in [KALW91] a simple sufficient and necessary feasibility test that can be applied if only two TT virtual links are to be scheduled:

$$C_i + C_k \leq \gcd(Tv_i, Tv_k) \tag{IV.4}$$

The following sections address the properties of scheduling multiple virtual links.

IV.4.3.2 Scheduling Multiple Tasks

The *sufficient and necessary* feasibility test presented by the Inequality (IV.4) does not hold for multiple tasks. Inequality (IV.5) presents a straight forward modification of that feasibility test which holds as a *sufficient* feasibility test.

$$\sum_{i=1}^{n} C_i \leq \gcd(\forall i, Tv_i) \tag{IV.5}$$

where n is the number of TT virtual links to be scheduled

This sufficient test is very restrictive. We refer the interested reader to [MS11], where the authors present a less restrictive sufficient test.

IV.5 Summary

TTEthernet is an extension of the switched Ethernet (IEEE 802.3) which allows for Best-Effort (BE), Rate-Constrained (RC) and Time-Triggered (TT) traffic to coexist in the same communication system In this chapter, we presented the fundamental properties of TTEthernet required to understand our scheduler proposed in Chapter V. Additionally, we present in this chapter the state-of-the-art in the generation of a schedule for TT virtual links in TTEthernet.

We started this chapter by presenting our terminology used to address the data transmission over TTEthernet. Then, we showed how the TTEthernet clock synchronization protocol impacts on the scheduling of TT-frames. In TTEthernet, scheduling a TT virtual link means reserving a time window (transmission windows) at a scheduled point in time for the transmission of each frame of that virtual link on each physical link in its path. In Section IV.3.2, we showed the similarities of the problem of scheduling TT-frames in a physical link and the problem of scheduling non-preemptive strictly periodic jobs in offset1 free systems (or independent operations strictly periodically).

A valid TT schedule for a given physical link shall not contain overlapping transmission windows, and therefore no contention among TT-frames. Nevertheless, contention

might occur due to the transmission of BE- and/or RC-frames, since their transmission times are not known in advance. We presented in Section IV.3.3 how TTEthernet addresses contention.

We depicted in Section IV.3.5 how the end-to-end latency of a frame varies according to the schedule on each physical link along the path of a virtual link, and presented how to reach the minimum latency. And in Section IV.3.6, we described a method to account for the temporal requirements of RC virtual links during the generation of a TT scheduling table.

Section IV.4 presented the state-of-the-art approaches in the generation of scheduling tables for TTEthernet.

Chapter V

Off-Line Scheduler for Time-Triggered Networks

This chapter presents an off-line scheduler for Time-Triggered Ethernet networks. Our scheduler, called Search Tree based Scheduler for Time-Triggered Networks (STSTTN), searches for a set of phases for the time-triggered traffic on each traversed physical link on the network, and ensures that the transmission time of time-triggered frames does not overlap. This scheduling problem is known to be NP-complete in the strong sense. Unless the optional *preventive tree pruning* approach is used, STSTTN is an optimal scheduler, in the sense that, if a message set is schedulable, the scheduler finds a feasible schedule. We start this chapter revisiting, in Section V.6, the reasons for the generation of scheduling tables in TTEthernet. Section V.2 describes how the TTEthernet clock synchronization protocol impacts on the scheduling table generation. We formalize the scheduling problem description in Section V.3. Section V.4 presents our proposed scheduler in detail and Section V.5 presents an empirical analysis of our scheduler based on an extensive set of experiments. Section V.6 concludes this chapter with a summary.

V.1 Motivation

Real-time distributed systems are connected by networks providing real-time properties. Such networks may apply two different paradigms for message transmission: event-triggered (ET) and/or time-triggered (TT). This chapter focuses on the latter.

Time-Triggered Ethernet is an extension to standard Ethernet that allows for the deterministic message transmission and offers both event-triggered and time-triggered message transmission (see Chapter I). For time-triggered transmissions, *"the sender and receiver(s) agree a priori on a cyclic time-controlled conflict free communication schedule for the sending of time-triggered messages"* [Kop11]. Therefore, each sender connected to a TT network requires a *scheduling table* containing the dispatching time of all frames[1] transmitted by this sender. Additionally, a scheduling table with the arrival time of frames, allows for a receiver to detect the late arrival of frames (or even missing frames) and react accordingly.

In TTEthernet, frames of same TT virtual link[2] are scheduled strictly periodically, i.e., the dispatching time of frames of the same virtual link is separated by equal time span. This chapter addresses the problem of generating a conflict free schedule for TT virtual links in all the traversed physical links of a TTEthernet network.

V.2 Impact of Clock Synchronization Protocol Frames

We describe in Section IV.3.1 the TTEthernet clock synchronization protocol including the relationship between protocol control frames (PCF), integration cycle and cluster cycle. Figure IV.3 summarizes this relationship. Notice that no frame used by the clock synchronization protocol (PCF) is transmitted after $t_{syncEnd}$ in each integration cycle. We assume that $t_{syncEnd}$ is known. Hence, no TT frame is scheduled between the start of an integration cycle and the respective $t_{syncEnd}$.

V.3 Scheduling Problem Formulation

We present in this section the formulation of the scheduling problem for TT virtual links in a TTEthernet network. Given are:

1. **Network topology ($\mathcal{T}$)** All connections between network nodes via unidirectional physical links in the network. It is represented via a digraph $\mathcal{T} = \langle Nodes, \mathcal{L} \rangle$, where $Nodes$ and $\mathcal{L}$ represent the set with all nodes and physical links, respectively.

2. **Virtual links ($\mathcal{V}^{TT}$, $\mathcal{V}^{RC}$)** Set with, respectively, all TT and RC virtual links in the network. Each TT and RC virtual link (v) defines its:

[1] Frame is the scheduled entity used to transmit messages. Figure IV.1 shows the relationship between *frames* and *messages*.

[2] Section IV.2 presents the definition of *TT virtual link*.

a) **Period (T_v)** Interval of time between the start of the transmission windows of two consecutive frames of an RC virtual link v. For RC virtual links, T_v represents the bandwidth allocation gap (BAG), i.e., the minimum time span between two consecutive frames of a TT virtual link, assuming no jitter.

b) **Transmission time (C_v)** Transmission window length of frames of virtual link v computed according to Equation (IV.1).

c) **Maximum Jitter ($Jmax_v$)** Maximum allowed delay imposed to a RC frame due to the dispatch of other frames at its source node. For TT frames, $Jmax_v = 0$.

d) **Path ($Path_v$)** Set of ordered physical links, elements of $\mathcal{L}$, traversed by virtual link v. If the virtual link v has multiple, e.g. np, destinations, we consider that one path for each destination exists ($Path_{v,1}, Path_{v,2}, \ldots, Path_{v,np}$). Each path $Path_{v,pa}$ is represented by $Path_{v,pa} = \{hop^1_{v,pa}, hop^2_{v,pa}, \ldots, hop^{nh}_{v,pa}\}$, where $\{hop^1_{v,pa}, hop^2_{v,pa}, \ldots, hop^{nh}_{v,pa}\}$ represents the ordered list of physical links from the source to the destination. We assume in this dissertation that the path (route) of each virtual link is also given as input.

V.3.1 Problem Statement

For every physical link in the network ($\forall\, l \in \mathcal{L}$), the TTEthernet scheduler shall compute a phase ϕ_v (see Section IV.2) for each traversing TT virtual link v such that no transmission windows overlap on that physical link.

V.3.2 Example

Let us consider an example with a set of virtual links as described in Table V.1 traversing one given physical link l. Notice that, the actual time unit of the task sets present in the next examples is irrelevant. Therefore, we consider that all time related values presented in all examples in this chapter, represent values in the same time unit. Figures V.1 and V.2 present a graphical representation of two sets of phases for virtual links: the first leading to an invalid and the latter to a valid schedule. Notice that the first three rows on these graphs show the schedule of individual TT virtual links (v_1, v_2, v_3) and the fourth row shows the complete physical link schedule. The schedule presented in Figure V.1 leads to an infeasible schedule: the 6th instance of v_1 collides with the 5th instance of v_3 while in the schedule presented in Figure V.2, no transmission windows overlap.

Table V.1: *Example virtual links set*

VL Id	T_v	C_v
1	10	1
2	60	1
3	12	1

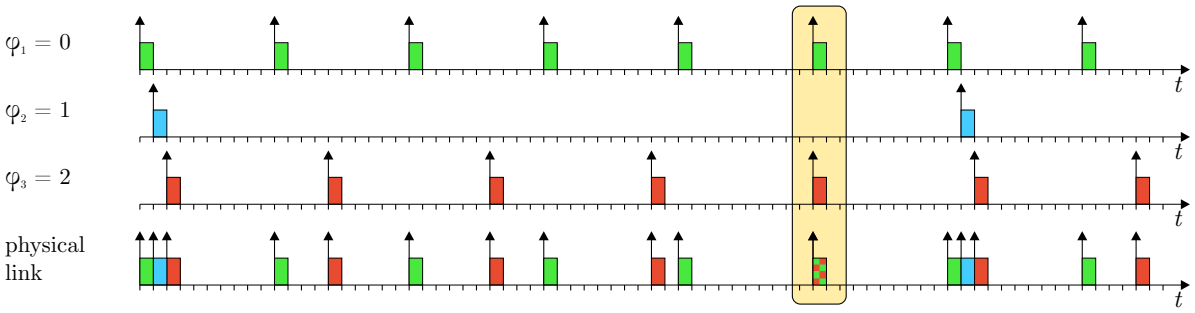


Figure V.1: *Invalid schedule. The 6th instance of v_1 collides with the 5th instance of v_3.*

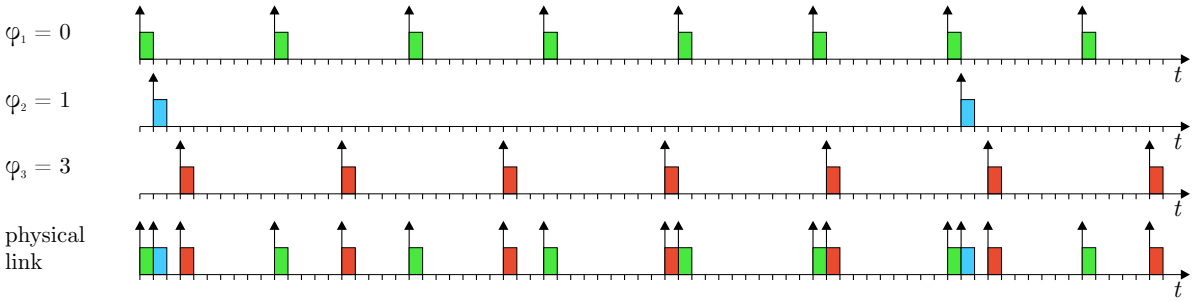


Figure V.2: *Valid schedule. No collision occurs*

Finding a valid schedule might be trivial if the set of virtual links traversing a physical link is small and their periods have special properties, e.g., harmonic periods, small least common multiplier (lcm). Otherwise, computing a valid schedule might require non-intuitive approaches.

V.4 Search Tree-based Scheduler for Time-Triggered Networks (STSTTN)

This section presents our solution for scheduling TT virtual links, called Search Tree-based Scheduler for Time-Triggered Networks (STSTTN). We refer to our solution, interchangeably, as *STSTTN* and *scheduler*. The core of STSTTN is a search tree, as described in Section V.4.2. STSTTN provides two (selectable) approaches to reduce the search space while traversing the search tree: *look back* and *look ahead*. Further, STSTTN allows for choosing multiple algorithms to select the edges while traversing the search tree. We present these approaches in Section V.4.3.

Section V.4.4 shows how to assign TT virtual links to tree levels. Section V.4.5 presents an heuristic to prune the search tree. Finally, we show in Section V.4.7 how to expand the scheduler in order to consider virtual links traversing multiple physical links. First we revisit the network assumptions used as input to STSTTN.

V.4.1 Network Assumptions

Section V.3 presents the set of TTEthernet properties related to the scheduling of TT virtual links. Next, we summarize the our network related assumptions.

- Frames of each TT virtual link v are transmitted strictly periodically, with period T_v.
- Transmission of a TT-frame is non preemptive.
- Time reserved for the transmission of each frame of each virtual link v is computed by Equation (IV.1).
- The time interval reserved for the clock synchronization protocol is not used to transmit TT-frames, i.e., in each integration cycle(see Section IV.3.1), the TT scheduler does not create any reservation window in the time interval from the start of an integration cycle until the arrival of the last protocol control frame ($t_{syncEnd}$) in that integration cycle.

V.4.2 Search Tree

We model the problem of scheduling virtual links traversing a physical link as a search tree. In this dissertation we use the terminology described by Cormen et al. in [CLRS01] to name the elements and properties of a rooted tree. Next, we present a summary of the relevant terms:

Parent, child: For any pair of vertices, e.g., x and y, if the last edge of x is (y, x), then y is a *parent* of x, and x is a *child* of y.

Ancestor: Any vertex on the unique simple path from the root until a vertex x is an *ancestor* of x.

Leaf: A vertex with no *child*.

Depth of a vertex: The length of the simple path from the root to this vertex.

Level: All vertices at the same *depth*. Thus, we name each level in a search tree by the corresponding depth.

Height of a vertex: Longest simple downward path from the vertex to a leaf.

Height of the tree: *Height* of the root vertex. Except if mentioned otherwise, in this chapter we use the term *height* to refer to the height of the root vertex.

Figure V.3 depicts a search tree for the set of TT virtual links presented in Table V.2. To improve readability, some vertices and edges do not explicitly appear in this (and next) figure(s) and are replaced by dots.

Table V.2: *Set with three virtual links*

VL Id	T_v	C_v
1	16	2
2	8	3
3	20	1

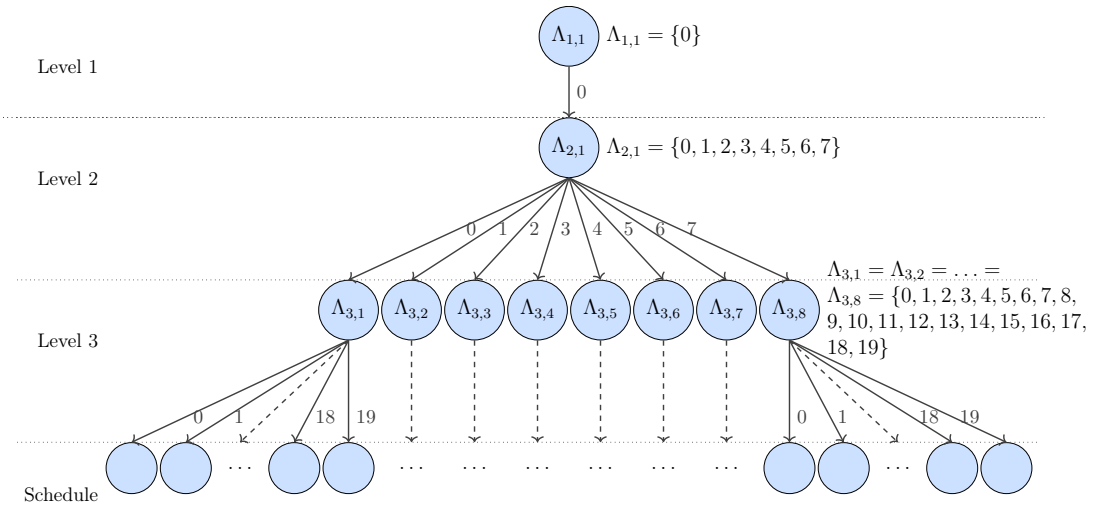


Figure V.3: *Complete enumeration of the search tree used to model the scheduling problem of the three TT virtual links presented in Table V.2*

Each *level* of the tree (except the deepest level) represents a virtual link. The example depicted in Figure V.3 has three levels representing virtual links, plus the "Schedule" level: in this example, virtual link ids correspond to tree levels.

Each *vertex* in a level represents a set of phases that can be used to schedule the corresponding virtual link. We denote each vertex (except those at the "Schedule" level) by the symbol $\Lambda_{l,c}$ where the index l represents the level, and c represents a counter on the number of vertices of that level, e.g., $\Lambda_{3,1}$ represents the first computed set of phases that can be assigned to the virtual link on level 3. Figure V.3 presents the elements of these sets.

Each *edge* represents a phase selected to schedule the corresponding virtual link. Notice that the value of an edge is equal to one element of the set represented by the parent vertex.

The *path* leading to each vertex describes a schedule, i.e., selection of phases, for the virtual links (represented by the tree levels) traversed by this path. The path to vertices in the "Schedule" level represent a complete schedule. Notice that also infeasible schedules are represented in this tree[3].

[3]This figure does not distinguish feasible from infeasible schedules.

The search tree in Figure V.3 shows the complete enumeration of vertices, i.e., all phases for each virtual link[4]. The total number of vertices in this example is equal to $(1+1+8+8\times 20 =)170$. The number of vertices grows exponentially with the period of each virtual link. Therefore, providing methods to reduce the search space drastically reduces computation time.

A simple approach to reduce the search space is to execute a feasibility test for each edge. If an edge surely leads to an infeasible schedule, then the scheduler deems this edge infeasible, discarding this edge and the underlying tree branch. Figure V.4[5] depicts the resulting search tree considering this simple approach. Notice that the total number of vertices is almost the half of the ones presented in the tree of Figure V.3, i.e., $(1 + 1 + 8 + 4 \times 20 =)90$. The applicability of such approaches depends on two conditions: the computational cost required to run the necessary feasibility test, and the amount of vertices that can be discarded in case the feasibility fails.

[4]Considering that one frame is transmitted per period, the largest valid phase value for a virtual link is equal to its period minus one (unit of time).

[5]Notice that in Figure V.4 and next figures, we mark with red color the vertices which, based on the current schedule, surely lead to infeasible schedules.

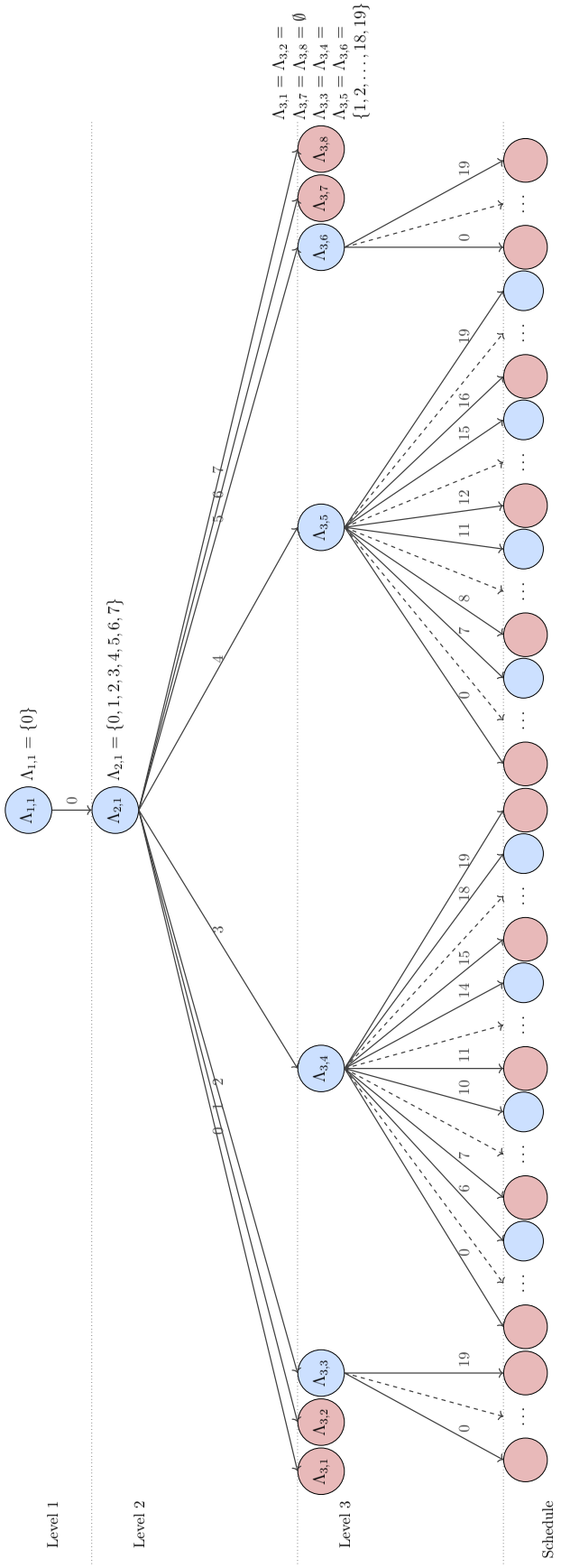


Figure V.4: *Search space reduction after applying a simple feasibility test per edge.*

Section V.4.3 shows how Equation (IV.3) provides a sufficient feasibility test with reduced computational cost. Not only does this approach lead to lower computational cost per schedulability test, but also to a smaller search space, when compared to the simpler approach presented in the previous paragraphs. For instance, applying this approach to the set of virtual links presented in Table V.2 leads to the search tree depicted in Figure V.5. Notice the decrease on the number of vertices (14 vs. 90), when compared to the search tree presented in Figure V.4.

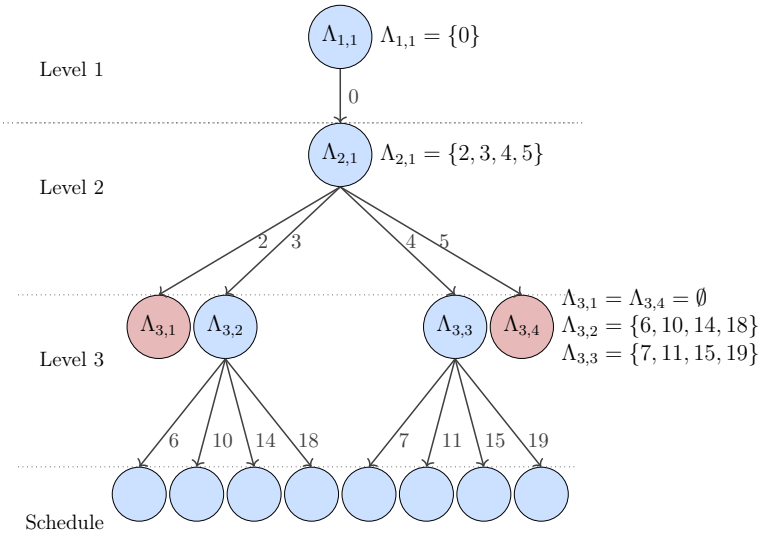


Figure V.5: *Reduced search tree using the approach presented in Section V.4.3.*

V.4.3 Traversing the Search Tree and Reducing Search Space

As STSTTN traverses a vertex, it selects one edge and computes the set of phases represented by the child vertex. The scheduler continues traversing the tree until it reaches a leaf. If STSTTN reaches a valid leaf, a feasible schedule is found and composed by the phases represented by the edges on the path from the root to the leaf.

An invalid leaf is a vertex with an empty set of feasible phases. If an invalid leaf is detected, the scheduler performs backtracking, i.e., revisits a valid ancestor vertex and selects a different edge. If no further backtracking is possible, e.g., the scheduler backtracks to the root node, the set of virtual links is deemed unschedulable. Section V.4.3.4 presents backtracking in details.

Considering the schedule of only *two virtual links* v_i and v_k, Equation (IV.3) provides the set Λ_k with all feasible phases for v_k assuming that v_i is scheduled at phase ϕ_i. One can consider this set as a kind of *sufficient* and *necessary* feasibility test, i.e., assuming that v_i is scheduled at phase ϕ_i, a feasible schedule for these two virtual links exists *if and only if* v_k is scheduled at one of the phases contained in the set Λ_k. If Λ_k is empty, then there exists no feasible schedule for v_i and v_k.

When scheduling multiple virtual links, the upper and lower bounds for feasible phases of each virtual link presented in Equation (IV.3) do not hold. Goossens shows in [Goo03] that, when scheduling multiple tasks in an offset free system, all non-equivalent choices for scheduling a task τ_i, considering $i-1$ tasks already scheduled, are represented by the half-open interval $[0, \gcd\{T_i, \text{lcm}\{T_1, \ldots, T_{i-1}\}\})$. Similarly, when scheduling a TT virtual link v_k, all non-equivalent choices for scheduling this virtual link are represented by the half-open interval $[0, \gcd\{T_k, \text{lcm}\{T_1, \ldots, T_{k-1}\}\})$. Next, we introduce Equation (V.1) which expands expands Equation (IV.3) to schedule multiple virtual links, by accounting for those bounds.

Assuming that $(k-1)$ TT virtual links have been scheduled before v_k, the set of feasible phases of v_k due to the schedule of v_i is given by:

$$\begin{aligned} &\Lambda_k = \phi_i + n \times \gcd(Tv_i, Tv_k) + m \\ &\forall n \in \mathbb{Z}, \forall m \in [C_i, \gcd(Tv_i, Tv_k) - C_k], \text{such that:} \\ &\quad \min(\Lambda_k) \geq 0, \max(\Lambda_k) < \gcd\left(T_k, \text{lcm}\{T_1, \ldots, T_{k-1}\}\right) \end{aligned} \tag{V.1}$$

When scheduling multiple virtual links, the sets computed by Equation (IV.3), comparing virtual links pairwise, do not provide a *necessary* and *sufficient* test anymore. Even though Equation (V.1) does not hold as *sufficient* test, the result obtained by this equation provides a *necessary* feasibility test: the "only if" condition of the pairwise computation still holds, i.e., assigning a phase which is not an element of Λ_k surely leads to an infeasible schedule.

Section V.4.3.1 presents the *look back* approach: an approach for computing $\Lambda_{l,c}$ at each level of the search tree considering the properties of the virtual links which have already been scheduled. Section V.4.3.2 presents an extension to this approach, called *look ahead*, which further accounts for the properties of virtual links that have not been scheduled yet.

V.4.3.1 Look Back Approach

Table V.3 shows an example with four TT virtual links. In this example, we assume that virtual link Ids represent their respective levels in the search tree. Next, we analyze how STSTTN traverses the search tree while reducing the number of vertices (w.r.t the complete enumeration). Figure V.6 depicts part of the search tree until the detection of an infeasible schedule and Table V.4 presents the computation of $\Lambda_{l,c}$ as the scheduler traverses the search tree.

Table V.3: *Example with four TT virtual links.*

VL Id	T_v	C_v
1	32	3
2	32	3
3	16	2
4	20	1

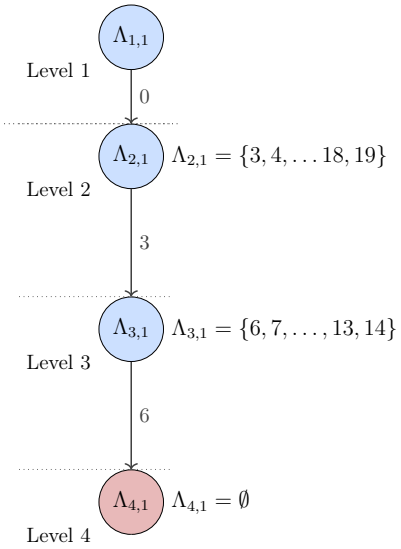


Figure V.6: *Partial schedule tree with look back.*

Table V.4: *Computation of* $\Lambda_{l,c}$ *using the* look back *approach.*

Edge	Pairwise Computation	Current Vertex
		$\Lambda_1 = \{0\}$
$\phi_1 = 0$	$\Lambda_{2vs.1} = \{3, 4, \ldots, 28, 29\}$	$\Lambda_{2,1} = \{3, 4, \ldots, 28, 29\}$
$\phi_2 = 3$	$\Lambda_{3vs.1} = \{3, 4, \ldots, 13, 14\}$ $\Lambda_{3vs.2} = \{6, 7, \ldots, 13, 14\}$	$\Lambda_{3,1} = \{6, 7, \ldots, 13, 14\}$
$\phi_3 = 6$	$\Lambda_{4vs.1} = \{3, 7, 11, 15, 19\}$ $\Lambda_{4vs.2} = \{2, 6, 10, 14, 18\}$ $\Lambda_{4vs.3} = \{0, 1, 4, 5, 8, 9, 12, 13, 16, 17\}$	$\Lambda_{4,1} = \emptyset$

The column *Edge* in Table V.4 presents the selected phase, at each level of the tree. In the column *Pairwise Computation*, Table V.4 shows the sets computed by Equation (V.1). Notice that we name $\Lambda_{k\,vs.\,i}$ the set of possibly feasible phases of v_k considering v_i scheduled at ϕ_i. The column *Current Vertex* shows the sets represented by the corresponding vertex ($\Lambda_{l,c}$), i.e., the intersection of the sets presented in the column *Pairwise Computation*.

STSTTN starts by assuming, without loss of generality, that v_1 is scheduled at 0, i.e., $\Lambda_1 = \{0\}$ and $\phi_1 = 0$. Then, STSTTN computes $\Lambda_{2vs.1}$, the set of feasible phases for v_2 considering that v_1 is scheduled at $\phi_1 = 0$. Considering that no other virtual link has been scheduled so far, the next vertex is represented by this set, i.e., $\Lambda_2 = \Lambda_{2vs.1}$. Let us assume that the scheduler selects $\phi_2 = 3$. Recall that, due to the "only if" (*necessary*)

property of the sets computed with Equation (V.1), phases for v_3 that are not elements of $\Lambda_{3vs.1}$ and $\Lambda_{3vs.1}$ surely lead to infeasible schedules. In other words, only phases present in the set computed by the intersection of $\Lambda_{3vs.1}$ and $\Lambda_{3vs.2}$ may lead to a feasible schedule. Thus, $\Lambda_{3,1} = \Lambda_{3vs.1} \cap \Lambda_{3vs.2}$. The computation of these intersections provide the elements of the sets presented in the column *Current Vertex*. The fourth row of Table V.4 shows the computation of $\Lambda_{4,1}$. Notice that the intersection $\Lambda_{4vs.1} \cap \Lambda_{4vs.2} \cap \Lambda_{4vs.3}$ results in an empty set, indicating that the phase selection $(\phi_1 = 0, \phi_2 = 3, \phi_3 = 6)$, represented by the path to this vertex, is infeasible. At this point, finding a feasible schedule requires *backtracking*, i.e., revisiting an ancestor vertex and selecting a different edge.

V.4.3.2 Look Ahead Approach

The look ahead approach uses Equation (V.1) to compute intermediate values of $\Lambda_{l,c}$ for the virtual links which have not been scheduled yet. The main goal in this approach is to detect infeasible phases as early as possible and thus decrease the search tree by not selecting edges that would lead to an invalid vertex at deeper levels of the search tree. Similar to the *look back* approach, after selecting a phase, the *look ahead* approach uses Equation (V.1) for the pairwise computation of feasible phases. However, *look ahead* computes the possibly feasible phases for all unscheduled virtual links.

We use the same example as in Section V.4.3.1 to present the *look ahead* approach. Table V.5 shows the resulting sets of each step in this approach and Figure V.7 depicts the part of the search tree until the detection of an infeasible schedule.

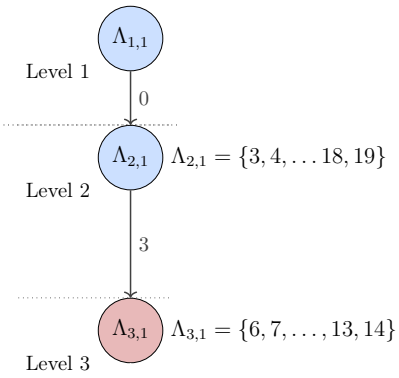


Figure V.7: *Partial schedule tree with look ahead.*

Table V.5: *Computation of* $\Lambda_{l,c}$ *using the* look ahead *approach.*

Edge	Pairwise Computation	Vertices Ahead	Current Vertex
			$\Lambda_{1,1} = \{0\}$
$\phi_1 = 0$	$\Lambda_{2vs.1} = \{3, 4, \ldots, 29\}$		$\Lambda_{2,1} = \{3, 4, \ldots, 29\}$
	$\Lambda_{3vs.1} = \{3, 4, \ldots, 14\}$	$\Lambda_{3,1} = \{3, 4, \ldots, 14\}$	
	$\Lambda_{4vs.1} = \{3, 7, 11, 15, 19\}$	$\Lambda_{4,1} = \{3, 7, 11, 15, 19\}$	
$\phi_2 = 3$	$\Lambda_{3vs.2} = \{6, 7, \ldots, 15\}$		$\Lambda_{3,1} = \{6, 7, \ldots, 14\}$
	$\Lambda_{4vs.2} = \{2, 6, 10, 14, 18\}$	$\Lambda_{4,1} = \emptyset$	

The scheduler selects $\phi_1 = 0$, for the same reasons as presented in Section V.4.3.1. It then computes the impact of ϕ_1 into the set of feasible phases of each unscheduled virtual link, i.e., $\Lambda_{2vs.1}, \Lambda_{3vs.1}, \Lambda_{4vs.1}$. Considering that only virtual link v_1 has been scheduled, $\Lambda_{2,1} = \Lambda_{2vs.1}$. We assume that the scheduler selects $\phi_2 = 3$. Then, the scheduler computes the impact of $\phi_2 = 3$ into the feasible phases of v_3 by computing the intersection of $\Lambda_{3,1}$ (presented in the *Vertex Ahead* column of the second row) and $\Lambda_{3vs.2}$. The resulting set is presented in the column *Current Vertex*. Similarly, the scheduler computes the impact of $\phi_2 = 3$ into $\Lambda_{4,1}$. Here, the intersection of $\Lambda_{4,1}$ (from previous row) and $\Lambda_{4vs.2}$ is an empty set, which in turn means that selecting $\phi_1 = 0, \phi_2 = 3$ leads to an infeasible schedule.

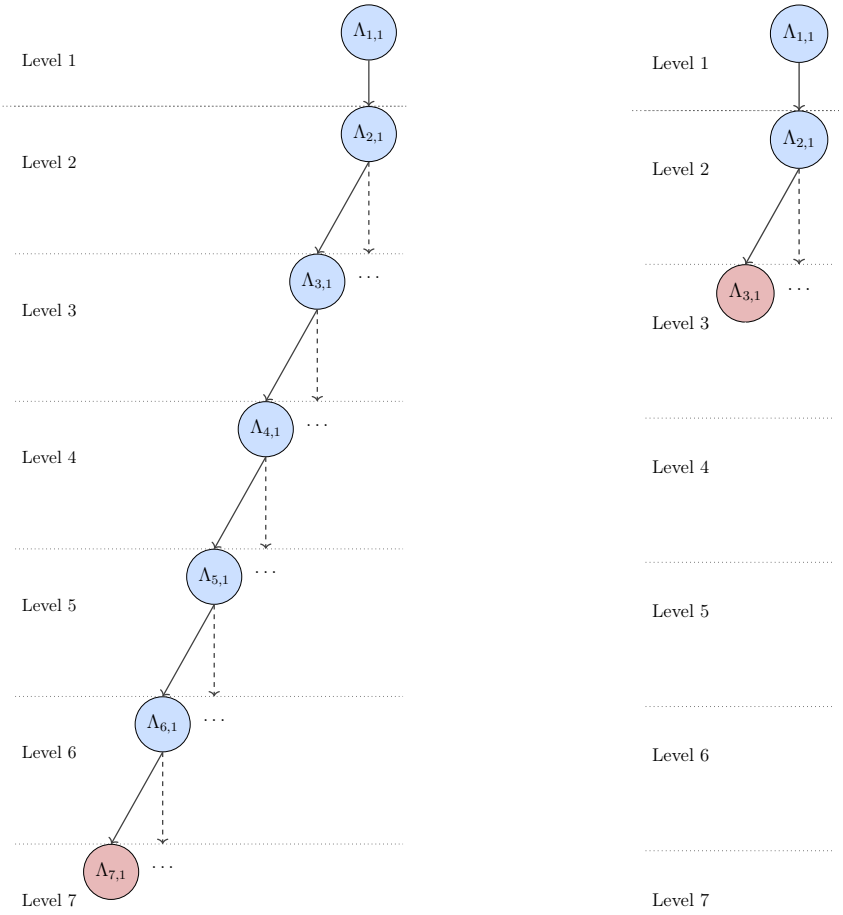
In comparison to the *look back* approach, the *look ahead* approach allows for detection of infeasible phases, at earlier levels of the search tree. Figures V.6 and V.7 show this property: the look back approach identifies an infeasible schedule after traversing 3 vertices while the look ahead approach identifies the same condition after traversing 2 vertices. Let us consider another example with 7 virtual links, as presented in Figure V.8. In this example, the selection of ϕ_3 leads to feasible phases for all virtual links except for v_7, i.e., $\Lambda_{7,1} = \emptyset$, independent of the selected phases ϕ_4, ϕ_5 and ϕ_6. For the *look back* approach, it takes the scheduler $(2 + 3 + 4 + 5 + 6 =)$ 20 pairwise computations (Equation (V.1)) to detect the phase selection that leads to the infeasible schedule. When applying the *look ahead* approach to the same set of virtual links, detecting the same condition takes $(6 + 5 =)$ 11 pairwise computations.

Comparing the numbers presented in the previous paragraphs, *look ahead* appears to be the preferable approach to be used by STSTTN. However, whenever an infeasible leaf is found, the scheduler must backtrack. Section V.4.6 shows that, in many cases, the computational backtracking cost is higher for *look ahead* than for *look back*.

Both *look back* and *look ahead* approaches assume that the assignment of virtual links to tree levels is given.

V.4.3.3 Selecting Edges

For a feasible set of virtual links, if the scheduler selects the *right* phases from the root to a valid leaf, a feasible schedule is found without any backtracking. Else, the scheduler might visit an enormous amount of vertices before it reaching a valid leaf. Therefore, the selection of edges in the search tree plays a major role. Figure V.9 shows two search



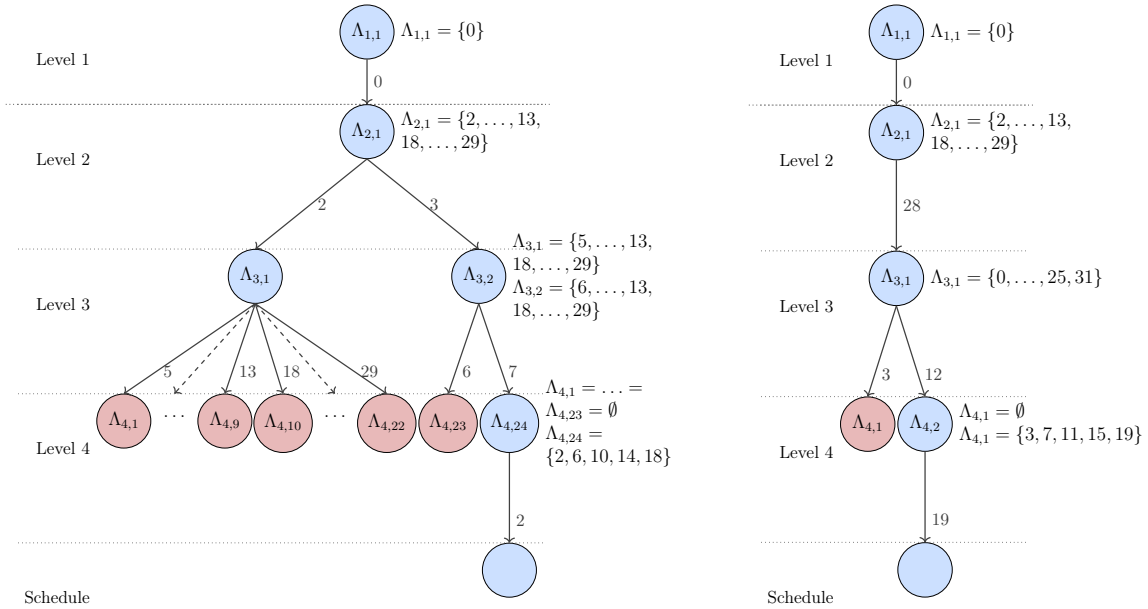
(a) *Look back* **(b)** *Look ahead*

Figure V.8: *Example comparing search trees with look back and look ahead.*

trees for the set of virtual links presented in Table V.6. In Figure V.9a, edges are selected sequentially from the smallest to largest element in the set of possibly feasible phases represented by the parent vertex. In Figure V.9b, edges are selected randomly from that set. In this example, selecting edges at random leads to a much smaller search tree. Nevertheless, this observation (random better than sequential) cannot be extrapolated to every set of virtual links. Section V.4.4 presents the effects on the search tree due to different edge selection methods.

Table V.6: *Example with same four TT virtual links presented in Sections V.4.3.1 and V.4.3.2. Here with other Ids, sorted by decreasing utilization.*

VL Id	T_v	C_v
1	16	2
2	32	3
3	32	3
4	20	1



(a) *Edges selected by increasing phases in* $\Lambda_{l,c}$ **(b)** *Edges selected randomly from phases in* $\Lambda_{l,c}$

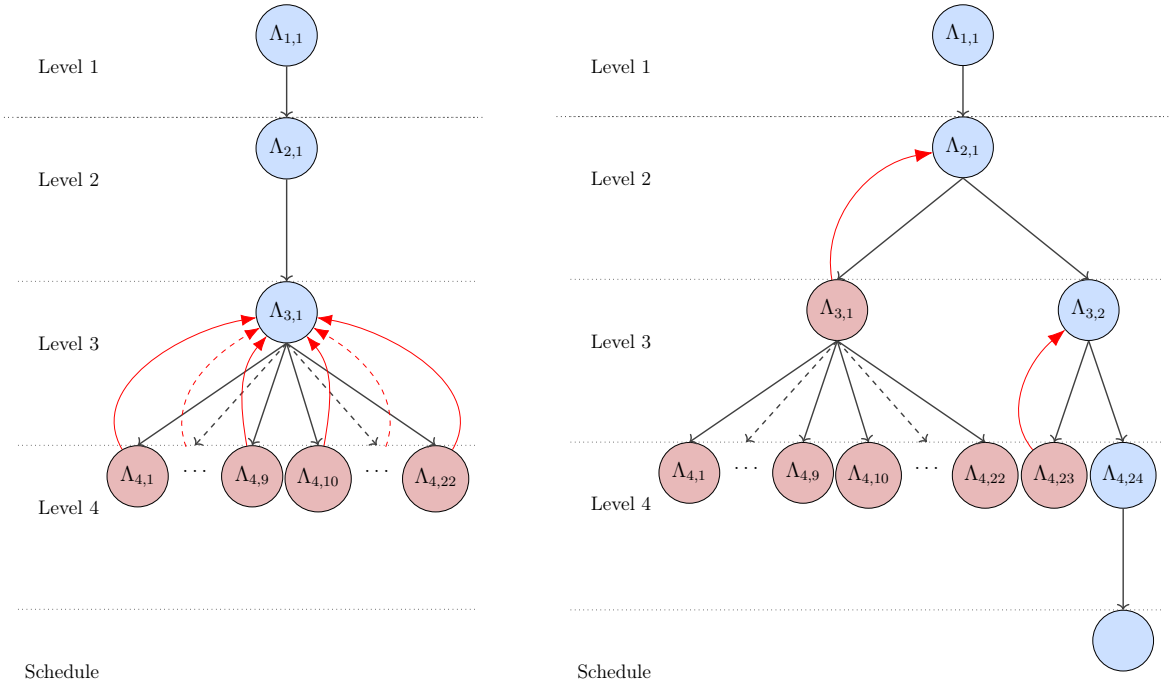
Figure V.9: *Search tree for two edge selection methods.*

V.4.3.4 Backtracking

After selecting an edge, STSTTN checks if the current path is feasible. Once STSTTN detects an invalid schedule, it discards the latest vertex on this path, revisits an ancestor vertex (usually the parent) and selects a different edge. Figures V.10a and V.10b show in details the backtracking of the example presented in Figure V.9a, in which the scheduler applies the *look back* approach. Remember that the index c of $\Lambda_{l,c}$ represents a counter for the number of vertices on a level of a tree, i.e., $\Lambda_{1,1}$ on one tree might be different from $\Lambda_{1,1}$ on another tree. Notice that the scheduler traverses the vertices $\Lambda_{1,1}, \Lambda_{2,1}, \Lambda_{3,1}$ until detecting an invalid vertex ($\Lambda_{4,1}$). After this point, the scheduler revisits $\Lambda_{3,1}$ and selects another edge, which again leads to an invalid vertex. After selecting all edges of $\Lambda_{3,1}$, the scheduler goes up another level and revisits $\Lambda_{2,1}$(see Figure V.10b). Again, the first edge leads to an invalid vertex ($\Lambda_{4,23}$) and to another backtrack. Finally, the scheduler reaches $\Lambda_{4,24}$ and after the next edge, it reaches a valid leaf, i.e., a valid schedule.

We discuss in Section V.4.5 a tree pruning heuristic to decrease the number of backtracks in the search tree.

Figure V.11 shows the search tree for an example with the same parameters as those of Figure V.10, except the scheduler applies the *look ahead* instead of the *look back* approach. Notice that no child vertex of $\Lambda_{3,1}$ leads to a feasible schedule. This condition is detected by the *look ahead* approach and consequently $\Lambda_{3,1}$ is deemed invalid. The scheduler revisits $\Lambda_{2,1}$, and selects another edge, which leads to an infeasible schedule, triggering another backtrack. The new path finally leads to a valid leaf, i.e., a valid schedule.



(a) *Backtrack from level four to level three*

(b) *Backtrack from level three to level two, and later from four to three.*

Figure V.10: *Backtracking with look back approach.*

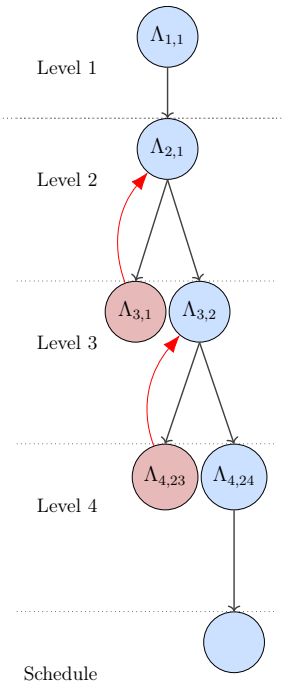
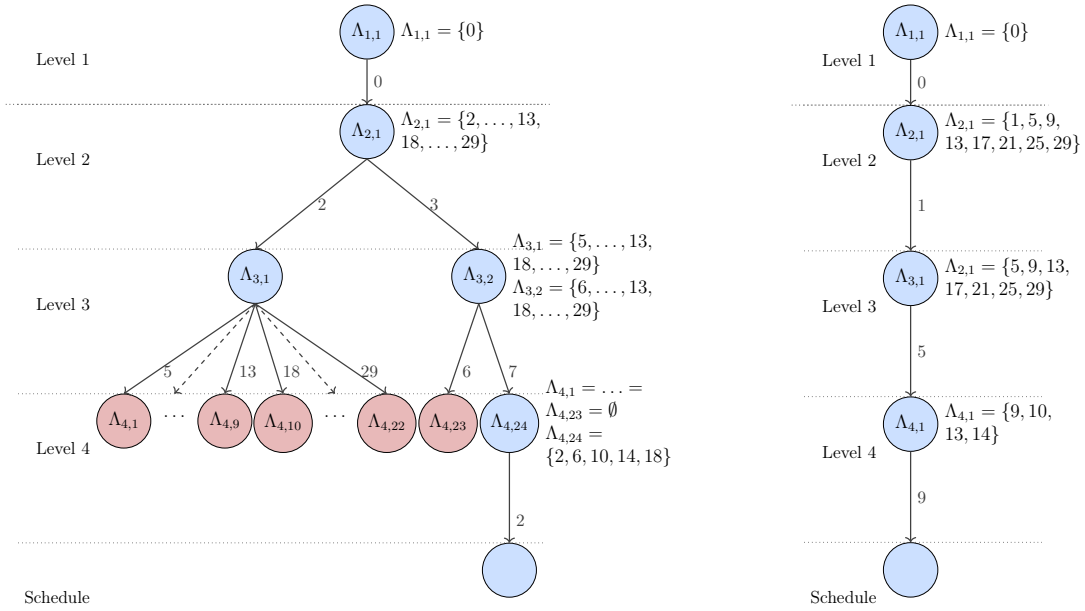


Figure V.11: *Backtracking with look ahead approach: backtrack from level three to level two, and later from four to three.*



(a) *Virtual links sorted by decreasing utilization* **(b)** *Virtual links sorted by increasing utilization*

Figure V.12: *Search tree for two assignments of virtual links to levels.*

V.4.4 Assigning Virtual Links to Levels

The assignment of virtual links to levels is static and defined before the scheduler starts traversing the tree. Depending on the method used for this assignment, a different search tree is constructed. Next we show how two assignment methods lead to different search trees and how these assignments impact the detection of feasible schedules. We use the same set of virtual links used in Sections V.4.3.1 and V.4.3.2. However, in order to improve readability, we change the virtual links Ids according to the assignment method, as presented in Table V.7. We assume in this section that STSTTN applies the *look back approach* and selects the edges by increasing order of phase.

Table V.7: *Example with same four TT virtual links presented in Sections V.4.3.1 and V.4.3.2. Here with other Ids, sorted by decreasing and increasing utilization, respectively.*

Decreasing Utilization				Increasing Utilization			
VL Id	T_v	C_v	U_v	VL Id	T_v	C_v	U_v
1	16	2	0.125	1	20	1	0.05
2	32	3	0.09375	2	32	2	0.0625
3	32	2	0.0625	3	32	3	0.09375
4	20	1	0.05	4	16	2	0.125

Figure V.15 depicts the search trees for virtual links sorted by decreasing and increasing utilization. Notice that the number of vertices in Figure V.12a (28 vertices) is more than five times larger than the one in Figure V.12b (5 vertices), i.e., considering the

Table V.8: *Example with four TT virtual links to depict the preventive tree pruning.*

VL Id	T_v	C_v
1	16	2
2	32	2
3	32	2
4	20	1

assumptions presented in the previous paragraphs, the scheduler finds a solution faster if virtual links are sorted by increasing utilization. Unfortunately, this observation does not hold for every set of virtual links, phase selection and virtual link sorting method. Section V.5 presents an empirical analysis accounting for different methods to assign virtual links to tree levels.

V.4.5 Preventive Tree Pruning

Examples presented in previous sections show that, depending on the properties of the virtual links and the scheduler parameters (assignment of virtual links to levels, edge selection, etc), many vertices on the search tree lead to infeasible schedules. Further, selecting a "bad" edge at upper levels of the tree might lead to infeasible schedules which can only be detected at deeper levels and consequently lead to a large number of backtracks before the scheduler deems the vertex invalid. Section V.4.3.2 presents the *look ahead* approach which addresses this issue and allows for earlier detection of an infeasible phase by accounting for the impact of scheduled virtual links on all unscheduled ones. Nevertheless, the impact of an unscheduled virtual link on another unscheduled one is not taken into account by the *look ahead* approach. We present in this section a preventive tree pruning heuristic, similar to the one presented in the Chapter 3 of Fohler's PhD Thesis [Foh94], to improve the early detection of invalid vertices. Table V.8 presents an example with a set of virtual links used to explain our preventive tree pruning approach.

Assuming that STSTTN applies the *look ahead* approach, selects the edges randomly and assigns virtual links to levels by Ids, Figure V.13 presents the corresponding search tree for $\phi_1 = 0$, $\phi_2 = 3$ and $\phi_2 = 4$ and all their children.

If the scheduler selects $\phi_2 = 3$, the probability to find a feasible schedule under the next edge is 0.43 ($\frac{10}{23}$). Else[6], if $\phi_2 = 4$, that probability is 0.78 ($\frac{18}{23}$). Consequently, STNTTS should select $\phi_2 = 4$ as soon as possible in order to increase the probability of finding a feasible schedule. The preventive tree pruning method addresses this issue by allowing for a maximum number of backtracks (a threshold called *branching factor* in [Foh94]) at each children of a vertex. Once this threshold is reached, the scheduler invalidates the current vertex and backtracks.

Figure V.14 depicts the effect of applying the preventive tree pruning heuristic with

[6]Remember that the scheduler computes the set of possibly feasible phases only after visiting the respective vertex. Consequently, these probabilities are not known until the scheduler visits the respective vertices.

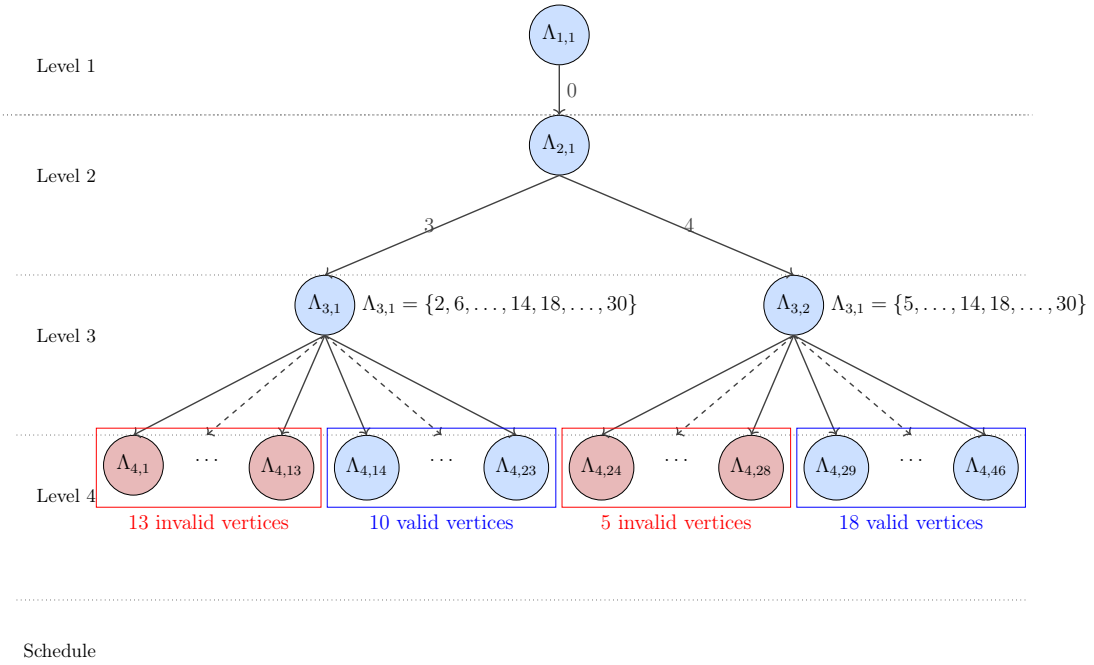


Figure V.13: *Search tree until level four.*

threshold equals to two.

Initially, the scheduler selects $\phi_1 = 0$ and $\phi_2 = 3$. After two backtracks in level four (from vertices $\Lambda_{4,1}$ and $\Lambda_{4,2}$), even though $|\Lambda_{3,1}| = 23$, the scheduler reaches the backtrack threshold and moves one extra level up in the tree (to $\Lambda_{2,1}$) and selects $\phi_2 = 4$. A feasible schedule is found once the scheduler reaches $\Lambda_{4,4}$, after one more backtrack.

As presented in the previous example, preventive tree pruning indirectly guides the scheduler to the vertices with higher probability of reaching a valid schedule. However, applying this heuristic eliminates the scheduler optimality, i.e., in the worst case, pruning the tree might remove all feasible schedules from the search space and the scheduler would not find a feasible schedule, even though one exists.

V.4.6 Traversing Costs

This section analyzes the computational costs for traversing the search tree according to the two approaches presented in Sections V.4.3.1 and V.4.3.2.

V.4.6.1 Look Back

Table V.4 shows the pairwise computations carried out while traversing the tree presented in Figure V.6. Notice that, at each level with depth d, the scheduler runs d pairwise computations and $d - 1$ intersections to compute the set of possibly feasible phases of the virtual link at that depth.

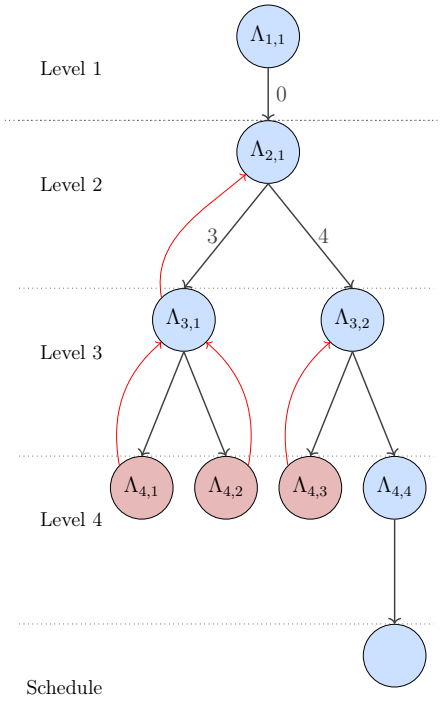


Figure V.14: *Search tree showing preventive tree pruning with threshold equals to two.*

Compared to the *downward* traversing cost, backtrack cost is not high: for each backtrack, the scheduler discards the edge leading to the invalid child vertex (at depth $d+1$), selects another edge from the upper depth and computes a new $\Lambda_{(d+1),x}$, where x represents a counter with the current number of vertices at depth $d+1$. Considering the number of computations per visited vertex, the closer to a leaf vertex the higher are the backtrack costs.

V.4.6.2 Look Ahead

In contrast to the *look back* approach, the cost of traversing vertices close to the root with the *look ahead* approach is high and decreases as the scheduler approaches the "Schedule" level. STSTTN executes $(height - d)$ pairwise computations and intersections per visited vertex at depth d (see Table V.5).

Backtracking also demands high computational costs at vertices close to the root: at depth d, backtracking requires the re-computation of the sets $\Lambda_{l,c}$ for all $(height - d + 1)$ unscheduled virtual links, i.e., the lower d the larger the amount of computations. Section V.5 presents an empirical analysis of the costs of *look back* and *look ahead* for different sets of virtual links and scheduler parameters.

V.4.7 Crossing Multiple Physical Links

In a simple approach, a schedule for the complete set of TT virtual links is obtained, if feasible, by running one instance of STSTTN for each physical link. Each of these instances, generates a schedule for all virtual links that traverse that physical link. A

feasible schedule for a given set of TT virtual links exists if STSTTN reaches a valid leaf for all physical links. Else, the set is deemed unschedulable. This approach, however may lead to large transmission latencies for the TT-frames: since the schedule of each physical link runs independently, in the worst case, the scheduled transmission of a frame in each physical link hop_h occurs a small amount of time before the scheduled frame arrival time. In this case, the latency increases by the time length of one period per traversed physical link.

V.4.7.1 Scheduling TT Virtual Links with Minimum Latency

In this section, we extend STSTTN to schedule the TT virtual links traversing the physical links of the network with minimal latency.

STSTTN starts by sorting all virtual links globally[7], i.e., applying the same algorithm that assigns virtual links to levels, to all virtual links. In the next steps, STSTTN creates one search tree for each physical link[8] on the network, and traverses these search trees. Once the scheduler reaches a valid leaf in all search trees, a valid schedule is found for every TT virtual link on the respective physical links. If on the contrary, the scheduler does not reach a valid leaf in a search tree, backtracking starts. In case STSTTN backtracks up to the root vertex of the search tree representing the source node of the first virtual link, this set of virtual links is deemed unschedulable.

The generated schedule allows for the transmission of TT frames with the minimum latency (see Section IV.3.5): the transmission window of a frame on a physical link starts as soon as this frame is ready for transmission. Figure V.15a depicts a schedule with this property: Notice that, after its scheduled arrival, the frame is ready for transmission sl (technological latency) units of time later. For the sake of simplicity and without loss of generality, we assume in the following examples that virtual links have only one destination and consequently one $Path_v$. For virtual links with multiple destinations, the same methods apply.

Next, we make use of an example to present how we extend STSTTN to allow for the scheduling with minimum latency. Let us assume that a virtual link v is scheduled at its source physical link with phase equals to $\phi_v^{hop_1}$. The phases scheduled on the three physical links in its path is depicted in Figure V.15a. In order to ensure the minimum latency of frames, the phase at any h-th physical link, in the path of a virtual link v, is computed by Equation (V.2).

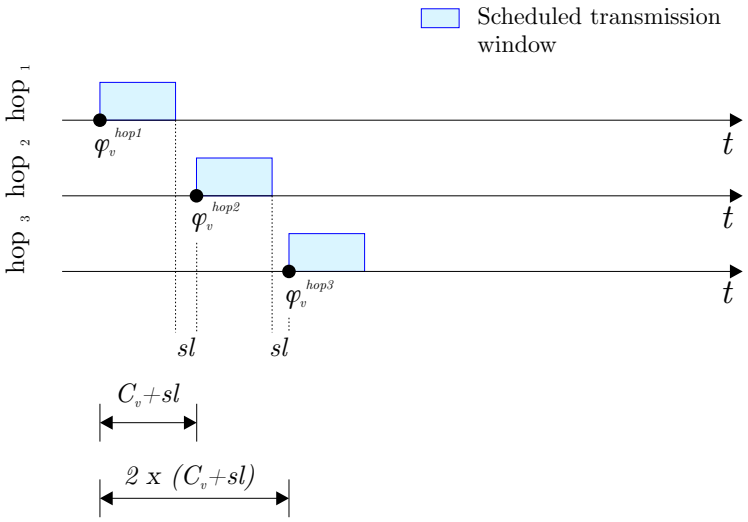
$$\phi_v^{hop_h} = \left(\phi_v^{hop_1} + (C_v + sl) \times (h-1)\right) \bmod T_v \tag{V.2}$$

where: sl represents the switching latency. Notice that the modulo function ensures that the value of $\phi_v^{hop_h}$ lays in the interval $[0, T_v)$.

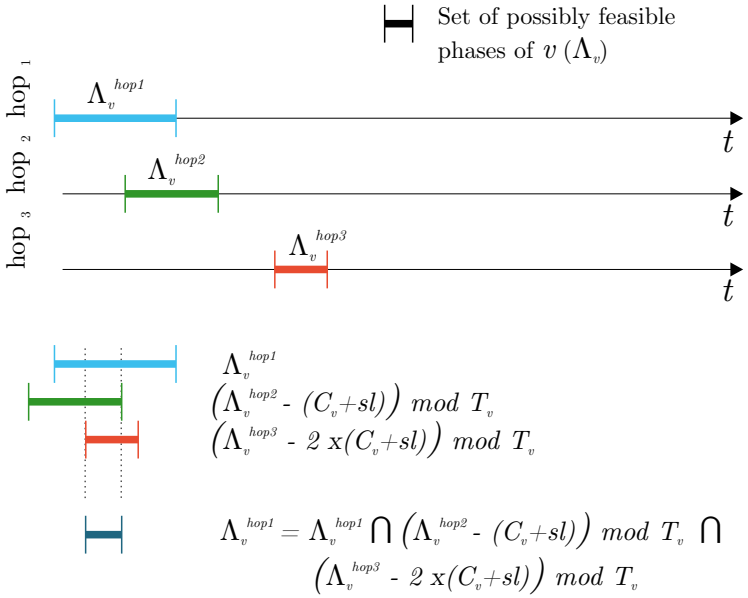
Next sections show how the *look back* and *look ahead* approaches address the scheduling for minimum latency problem on multiple search trees. For the sake of simplicity,

[7]Sorting the virtual links globally ensures that they are sorted by the same algorithm also within each search tree.

[8]Recall that each bi-directional network connection is modeled as two unidirectional physical links



(a) *Scheduled phases on each physical link in the path of the virtual link v.*



(b) *Computation of the updated value for $\Lambda_v^{hop_1}$.*

Figure V.15: *Relationship between Λ and latency along the path of the virtual link v.*

we omit in the next sections the counter index from the notation of the sets of possible feasible phases, e.g., $\Lambda_v^{hop_h}$ instead of $\Lambda_{v,1}^{hop_h}$.

V.4.7.1.1 Look Back We continue the analysis of the virtual link v, traversing the physical links in its path $Path_v = hop_1, hop_2, hop_3$. The scheduler computes the set of possibly feasible phases for v on each physical link along its path $(\Lambda_v^{hop_1}, \Lambda_v^{hop_2}, \Lambda_v^{hop_3})$ using the same method as presented in Section V.4.3.1. Notice that these sets account only for the impact of the already scheduled virtual links in each search tree.

In order to account for the relationship between transmission windows along the virtual link path, STSTTN "integrates" the sets $\Lambda_v^{hop_1}, \Lambda_v^{hop_2}, \Lambda_v^{hop_3}$ into an updated set

$\Lambda_v^{hop_1}$. This updated $\Lambda_v^{hop_1}$ provides the set of possibly feasible phases of v in its source physical link.

Let us analyze the impact of $\Lambda_v^{hop_2}$ into $\Lambda_v^{hop_1}$. Recall that $\Lambda_v^{hop_2}$ represents the set of possibly feasible phases of v in the physical link hop_2. Thus, the virtual link v might only be scheduled in hop_2 if a phase π_2 exists such that $\phi_v^{hop_2} = \pi_2 \in \Lambda_v^{hop_2}$. However, considering that the transmission window starts immediately after the frame arrival, this schedule depends on the schedule of v in the source physical link. More specifically, due to the transmission delay from hop_1 to hop_2, virtual link v must be scheduled at its source physical link at a phase $\phi_v^{hop_1} = \pi_1 = \big(\pi_2 - (C_v + sl)\big) \bmod T_v$. Phase π_1 is valid only if $\pi_1 \in \Lambda_v^{hop_1}$. Considering only the two initial physical links in the path of v, we conclude that $\phi_v^{hop_1} = \pi_1$ is a feasible phase only if $\pi_1 = \big(\pi_2 - (C_v + sl)\big) \bmod T_v \in \Lambda_v^{hop_1}$ and $\pi_2 \in \Lambda_v^{hop_2}$, i.e., $\pi_1 \in \Lambda_v^{hop_1} \cap \big(\Lambda_v^{hop_2} - (C_v + sl)\big) \bmod T_v$. Figure V.15b shows the sets $\Lambda_v^{hop_1}, \Lambda_v^{hop_2}, \Lambda_v^{hop_3}$, and the updated set $\Lambda_v^{hop_1}$ after accounting for the latency on each physical link.

Equation V.3 presents the generalization of this analysis for a path with multiple (nh) physical links providing the updated value for $\Lambda_v^{hop_1}$.

$$\Lambda_v^{hop_1} = \Lambda_v^{hop_1} \cap \Big(\Lambda_v^{hop_2} - (C_v + sl)\Big) \bmod T_v \cap \Big(\Lambda_v^{hop_3} - 2 \times (C_v + sl)\Big) \bmod T_v \cap \ldots \\ \cap \Big(\Lambda_v^{hop_{nh}} - (nh - 1) \times (C_v + sl)\Big) \bmod T_v \tag{V.3}$$

If the set resulting from the computation of Formula (V.3) is not empty, then the scheduler inserts the vertices $\Lambda_v^{hop_1}, \Lambda_v^{hop_2}, \ldots, \Lambda_v^{hop_{nh}}$ in the respective search trees, and continues traversing the search trees. Else, the vertices $\Lambda_v^{hop_1}, \Lambda_v^{hop_2}, \ldots, \Lambda_v^{hop_{nh}}$ are deemed invalid and the scheduler backtracks. Section V.4.7.1.3 presents backtracking in detail.

V.4.7.1.2 Look Ahead The computation of Λ presented in Section V.4.3.2 accounts not only for the properties of scheduled virtual links, but also for those of not yet scheduled virtual links: After selecting a phase, the scheduler computes the intermediate Λ (called *Vertices Ahead* in Table V.5) for each unscheduled virtual link in each tree of $Path_v$. Extending the approach presented in that section to a account for the relationship between the transmission windows along the virtual link path, is done applying the same method presented in Section V.4.7.1.2: STSTTN updates the set $\Lambda_v^{hop_1}$ applying Formula (V.3). Notice that the sets $\Lambda_v^{hop_1}, \Lambda_v^{hop_2}, \Lambda_v^{hop_3}, \ldots, \Lambda_v^{hop_{nh}}$ represent the *Current Vertex* sets computed for the other trees.

If any intermediate value of Λ, *Current Vertex*, or the updated set $\Lambda_v^{hop_1}$ is empty, the vertices $\Lambda_v^{hop_1}, \Lambda_v^{hop_2}, \Lambda_v^{hop_3}, \ldots, \Lambda_v^{hop_{nh}}$ are invalidated and the scheduler backtracks (see Section V.4.7.1.3). Otherwise, the scheduler inserts these vertices into the respective search trees, and continues traversing the search trees.

V.4.7.1.3 Backtracking Section V.4.3.4 presents backtracking assuming that all virtual links traverse one physical link. According to the description in that section, when backtracking, the scheduler invalidates the current vertex (corresponding to the virtual

link to be scheduled), revisits one ancestor vertex and selects another edge. However, considering that virtual links traverse multiple physical links, two questions arise.

Which vertices should be invalidated? In order to maintain a coherent schedule state among all search trees, the scheduler invalidates the vertices corresponding to the virtual link that triggered the backtrack in every search tree representing the physical links on the path of this virtual link.

Which vertices should be revisited? Revisiting the latest scheduled virtual link (latest added vertex), might lead to the exact same invalid vertices that triggered the backtracking, if the path of the last scheduled virtual link and the virtual link of the invalid vertex do not cross (directly or indirectly - see below). In order to identify the vertices candidate to be revisited, we describe next three types of relationship between virtual links and explain how they impact on the choice of the vertex to be revisited in case of backtracking.

Direct crossing We say that a virtual link v_a directly crosses a virtual link v_b if any physical link in $Path_{v_a}$ is also present in $Path_{v_b}$, i.e., if their paths overlap.

Indirect crossing We say that a virtual link v_a indirectly crosses a virtual link v_b if, even though the path of these two virtual links do not overlap, the schedule of v_a might impact the schedule of v_b.

No crossing We say that a virtual link v_a does not cross a virtual link v_b if the schedule of v_a does not impact the schedule of v_b.

The scheduler selects the latest virtual link (v_b) that directly or indirectly crossed the virtual link (v_b) of the invalid vertex, and revisits the corresponding vertex of v_a in each search tree on its path. Finally, the scheduler selects another phase in the source tree of the revisited virtual link.

In summary, once a vertex is deemed invalid, the scheduler backtracks executing three steps. Assuming STSTTN finds an invalid vertex $\Lambda_{v_a}^{hop_1}$, then STNTTS:

1. invalidates the vertices corresponding to v_a in all trees $(\Lambda_{v_a}^{hop_1}, \Lambda_{v_a}^{hop_2}, \ldots, \Lambda_{v_a}^{hop_{nh}})$
2. selects the latest direct or indirect crossing virtual link, e.g., v_b
3. selects another edge from $\Lambda_{v_b}^{hop_1}$

This strategy does not jeopardize the scheduler optimality, i.e., no valid schedule is discarded.

V.5 Evaluation

Our experiments depict the performance of STSTTN when scheduling virtual links traversing one physical link. We do not address here the schedule of multiple physical links. Next we present how we generate the set of virtual links used in the experiments.

V.5.1 Generator of Virtual Links Set

We use our Virtual Links Set Generator (VLSG) to create the sets of virtual links used as input to STSTTN. VLSG creates a set of virtual links according to the following input parameters:

1. number of virtual links,
2. target utilization,
3. maximum utilization error,
4. minimum and maximum C per virtual link,
5. minimum and maximum period per virtual link, and
6. the base and exponent values used to compute the virtual link periods (explained next).

In a first step, VLSG creates an array with a period for each virtual link. In order to keep control over the least common multiplier of all periods, we use a modified version the method presented in [GM01] to compute the period of the virtual links. Each period is the result of a multiplication of factors obtained from the elements of a matrix, where rows represent the bases, and the columns represent the respective exponent of a base (see matrix M). In the method presented in [GM01] bases are prime numbers. In VLSG, we relax this restriction. Next is an example of such a matrix.

$$M = \begin{pmatrix} 0 & 1 & 1 & 2 & 3 & 7 \\ 1 & 2 & 2 & 2 & 4 & 5 \\ 0 & 1 & 2 & 4 & 6 & 8 \end{pmatrix} \begin{matrix} \text{Base 2} \\ \text{Base 3} \\ \text{Base 5} \end{matrix}$$

Each factor is calculated by randomly selecting a column representing the exponent for each base. Then, the period is computed by multiplying all factors. For instance, selecting columns 5, 1, 3 for rows 1, 2, 3 respectively, results in a period of: $2^3 \times 3^1 \times 5^2 = 600$. If a computed period is not within the boundaries defined by minimum and maximum period, it is discarded and recomputed.

We can influence the probability of selecting a given exponent by increasing the number of times that this exponent appears in the matrix. For instance, the probability to select the exponent 2 is three times larger than exponent 4 for base 3.

In a second step, VLSG applies the *UUnifast* [BB05] algorithm to ensure that the utilization of the set of virtual links matches the specified target utilization. Further, *UUnifast* ensures a uniform distribution among utilizations to each virtual link in the set.

In a third step, VLSG computes C (transmission window length) for each virtual link, such that the utilization computed in the second step is achieved using the period computed in the first step. If the computed C is not within the boundaries defined by minimum and maximum C, then the set of virtual links is considered valid, else VLSG

restarts from the first step. Notice that, due to non deterministic decisions taken in the first and second step, restarting the generation of the set of virtual links might lead to a valid set.

Finally, VLSG computes the difference between the actual utilization (sum of the utilization of each virtual link) and the target utilization. If this value is lower than the input parameter *maximum utilization error*, then the set of virtual links is considered valid, else VLSG restarts from the first step.

Recall that a necessary and sufficient schedulability test for a set of TT virtual links is a NP-complete problem. Therefore, even though the total utilization of the virtual link sets generated by VLSG is less than one, these sets are not guaranteed to be feasible.

V.5.2 Experiments

We implemented STSTTN in a single threaded C++ application and executed the experiments described in this section in the *high performance computer Elwetritsch* at the Technische Universtät Kaiserslautern. Our experiments ran on a node with Intel Xeon E5 2670 processor at 2.6GHz and 32Gb of RAM.

For each experiment, we select a group of sets of virtual links as described in Table V.9, and one (or more) set(s) of configuration parameters for STSTTN, as described in Table V.10. The properties shown in Table V.10 describe the name of a virtual link set, the number of virtual link in the set, the utilization, the matrix used to generate the periods, and transmission window boundaries for each virtual link. Notice that we provide explicit lower and upper bounds for the periods (besides the matrix of bases and exponents).

For each run, we set a timeout of one hour, i.e., we abort the execution of STSTTN if no feasible schedule is found within this timespan. The name of each experiment is formed by concatenating the name of the group of virtual link sets, and the STSTTN configuration, e.g. Cra1u5_C1C2C3C4.

Table V.9: *Input to the scheduler: set of virtual links*

Name	# VLs	U	Matrix Period	C_{min}	C_{max}
AFDXu8v200	200	0.8	M_{A1}	2	122
AFDXu9v200	200	0.9	M_{A1}	2	122
AFDXu9v400	400	0.9	M_{A1}	2	122
Cra1u5	10	0.5	M_{C1}	2	9
Cra1u8	10	0.8	M_{C1}	2	9
Cra1u9	10	0.9	M_{C1}	2	9
Cra2u5	10	0.5	M_{C2}	2	9
Cra2u7	10	0.7	M_{C2}	2	9
Cra2u8	10	0.8	M_{C2}	2	9
Cra2u9	10	0.9	M_{C2}	2	9

$$M_{A1} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & & & & & & & \end{pmatrix} \begin{matrix} \text{Base 2} \\ \text{Base 10} \end{matrix}$$
$$T_{A1min} = 1000, T_{A1max} = 128000$$
$$M_{C1} = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 2 & \end{pmatrix} \begin{matrix} \text{Base 2} \\ \text{Base 5} \end{matrix}$$
$$T_{C1min} = 10, T_{C1max} = \text{unbounded}$$
$$M_{C2} = \begin{pmatrix} 0 & 1 \\ 1 & 2 \end{pmatrix} \begin{matrix} \text{Base 3} \\ \text{Base 10} \end{matrix}$$
$$T_{C2min} = 10, T_{C2max} = 100$$
$$M_{C3} = \begin{pmatrix} 0 & 1 \\ 1 & 2 \\ 0 & 1 \end{pmatrix} \begin{matrix} \text{Base 3} \\ \text{Base 5} \\ \text{Base 10} \end{matrix}$$
$$T_{C3min} = 50, T_{C3max} = 75$$

Table V.10: *Input to the scheduler: set of configuration parameters*

Name	VL to Tree Level	Traversing Approach	Edge Selection
CONF1	U Descending	Look Back	Sequential
CONF2	U Descending	Look Back	Random
CONF3	U Descending	Look Ahead	Sequential
CONF4	U Descending	Look Ahead	Random

The goals of the experiments is to test the following hypothesis:

H1. *Look ahead* is more efficient than *look back*, i.e., it requires less backtracking and consequently shorter time to compute a feasible schedule or to detect infeasibility using *look ahead* instead of *look back*.

H2. For a similar set of periods, the higher the utilization of the virtual link set, the harder it is for STSTTN to find a feasible schedule.

H3. The algorithms to assign virtual links to levels of the search tree, and to select the edges have a large impact on the time required to compute a feasible schedule and/or to detect infeasibility.

H4. STSTTN can rapidly compute a feasible schedule for a set of virtual links with harmonic periods. The same applies for sets in which the smallest period is equal to the greatest common divisor of all virtual link periods.

We start presenting the results collected from experiments using virtual link sets of the *super group Cra*. These sets are based on the three set of periods presented in [CO16]. For each group of periods, we generated generate virtual link sets with utilization varying from 0.5 until 0.9. We generated 500 sets of 10 virtual links for

each combination of utilization and group of periods (see Table V.9 from Cra1u5 to Cra3u9). For all experiments of the *super group* Cra, we execute STSTTN with the configuration parameters CONF1, CONF2, CONF3, and CONF4. For all experiments of the *super group* AFDX, we execute STSTTN with the configuration parameters CONF1, and CONF3. Notice that, boxplots from configurations which have not been executed, present -1 as result, e.g., Figure V.24d.

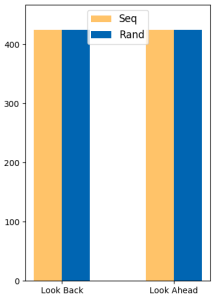
For all experiments of the *super group* Cra, we can observe that *look ahead* performs better than *look back*, i.e., when compared to *look back*, *look ahead* requires a smaller number of backtracks to compute a feasible schedule or to detect infeasibility. Figure V.21 shows that no set of virtual links requires more than 20 backtracks (3 seconds) to compute a feasible schedule and no more than 800 backtracks (1 second) to detect an infeasible set using *look ahead* with sequential selection of edges. Whereas *look back* requires up to 300.000 backtracks (200 seconds) and up to 6.000.000 backtracks (3500 seconds) to achieve the same conclusion.

The number of backtracks presented in Figures V.19, V.20, V.21,and V.22, increase with the utilization of the set of the virtual link, irrespective to the approach used to traverse the search-tree. Notice that these numbers increase for both computing a feasible schedule and to deem a set of virtual links infeasible.

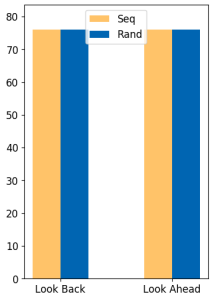
The experiments conducted for the *super group* Cra show that, in comparison to the sequential selection of phases, selecting edges (phases) randomly is disadvantageous if virtual links are assigned to the levels of the search tree by decreasing order of utilization.

We observed, in another set of experiments, that assigning virtual links to the levels of the search tree by increasing order of utilization, drastically reduces the schedulability ratio a set of virtual links, i.e., a timeout occurs before STSTTN can find a feasible schedule. Therefore we do not present those experiments here.

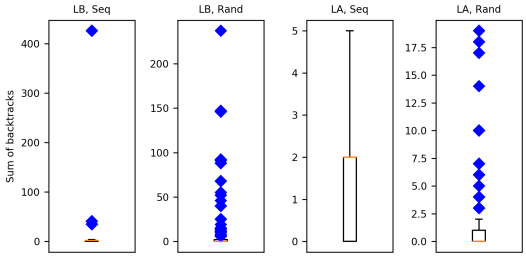
In the *super group* afdx, the periods of the virtual links are elements of the allowed BAG values of AFDX networks, i.e., 1, 2, 4, 8, 16, 32, 64 and 128 ms, and the length of their transmission windows are equivalent to the transmission time of frames in AFDX networks at 100Mbps. Due to the harmonic characteristic of the periods in these sets, STSTTN can compute feasible schedules in short time even for large utilizations. Figure V.26g depicts for instance that both *look back* and *look ahead* approach can compute a feasible schedule for a set with utilization of 0.9 in up to 130 seconds and 80 seconds, respectively, without backtracks.



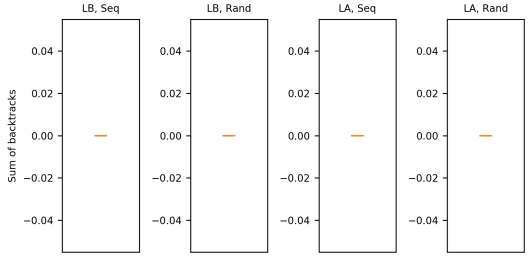
(a) *Number of scheduled sets.*



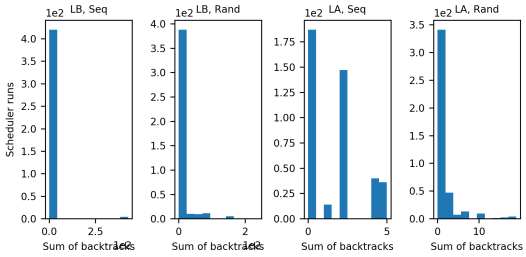
(b) *Number of not scheduled sets.*



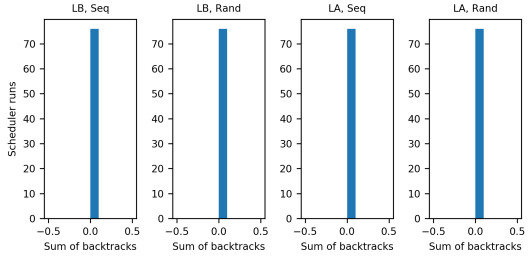
(c) *Boxplot with sum of backtracks per generated schedule.*



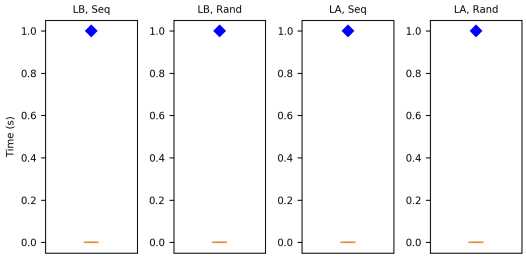
(d) *Boxplot with sum of backtracks until deem unschedulable.*



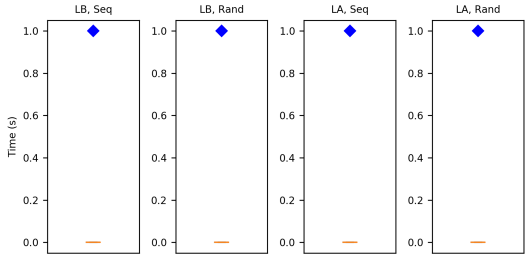
(e) *Histogram with sum of backtracks per generated schedule.*



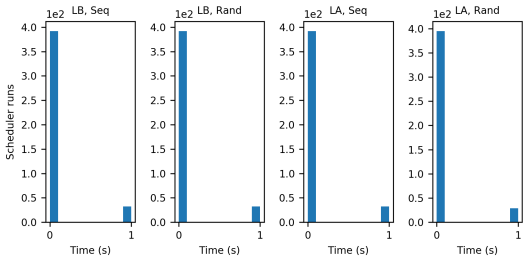
(f) *Histogram with sum of backtracks until deem unschedulable.*



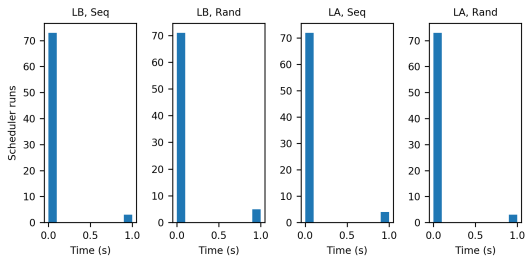
(g) *Boxplot with time to generate a schedule.*



(h) *Boxplot with time to deem unschedulable.*

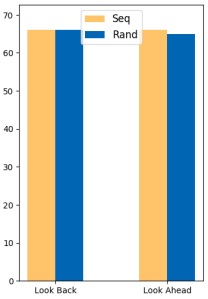


(i) *Histogram with time to generate a schedule.*

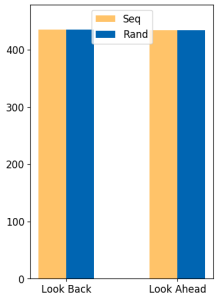


(j) *Histogram with time to deem unschedulable.*

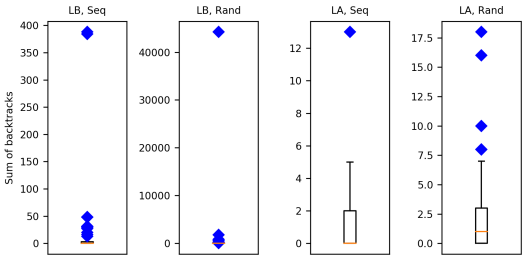
Figure V.16: *Experiment Cra1u5: STSTTN runs for 500 sets of virtual links. Sets scheduled by STSTTN on the left and not scheduled on the right. Data classified by traversing and edge selection algorithms.*



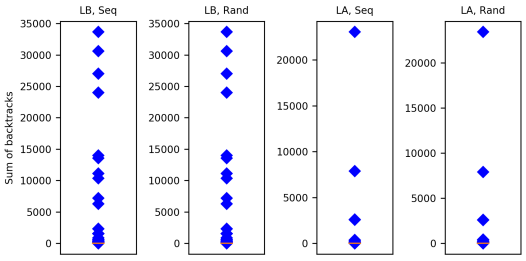
(a) *Number of scheduled sets.*



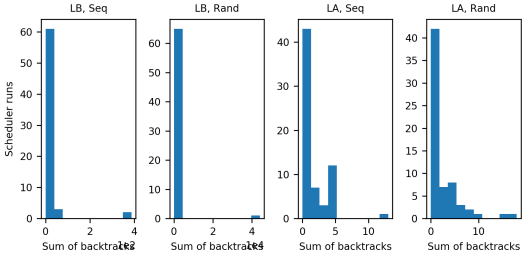
(b) *Number of not scheduled sets.*



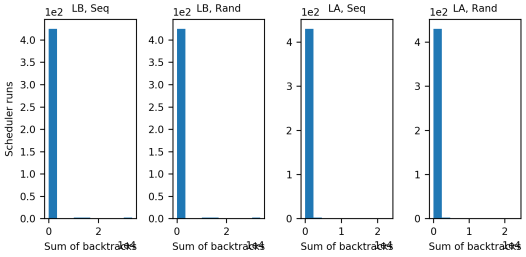
(c) *Boxplot with sum of backtracks per generated schedule.*



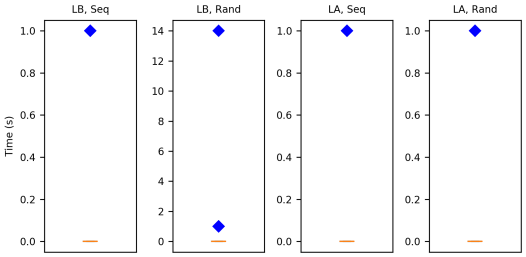
(d) *Boxplot with sum of backtracks until deem unschedulable.*



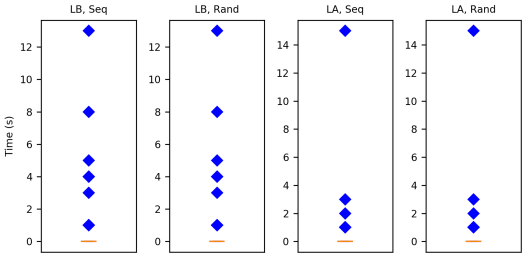
(e) *Histogram with sum of backtracks per generated schedule.*



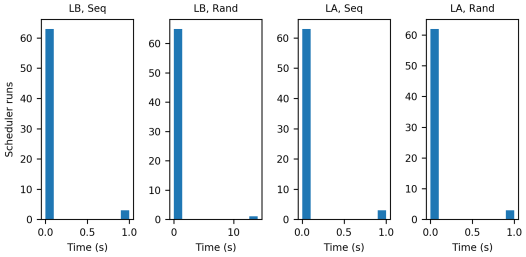
(f) *Histogram with sum of backtracks until deem unschedulable.*



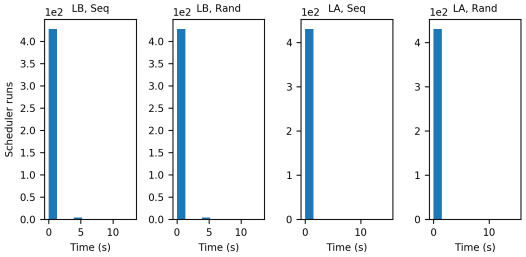
(g) *Boxplot with time to generate a schedule.*



(h) *Boxplot with time to deem unschedulable.*

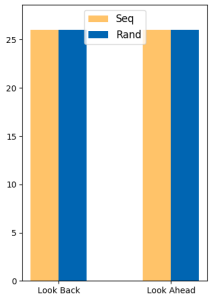


(i) *Histogram with time to generate a schedule.*

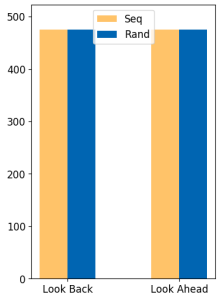


(j) *Histogram with time to deem unschedulable.*

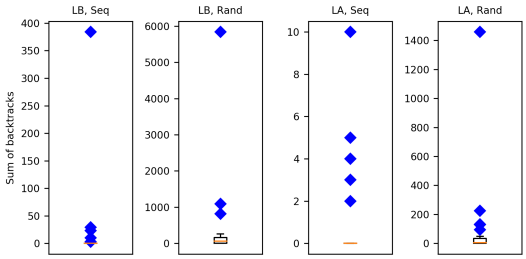
Figure V.17: *Experiment Cra1u8: STSTTN runs for 500 sets of virtual links. Sets scheduled by STSTTN on the left and not scheduled on the right. Data classified by traversing and edge selection algorithms.*



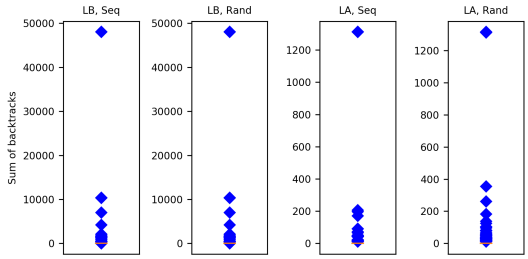
(a) *Number of scheduled sets.*



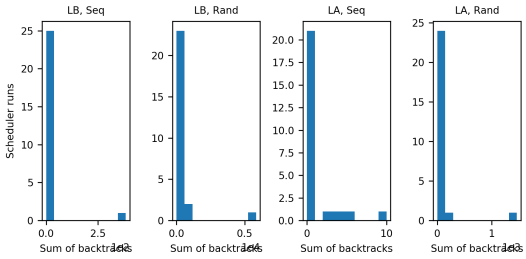
(b) *Number of not scheduled sets.*



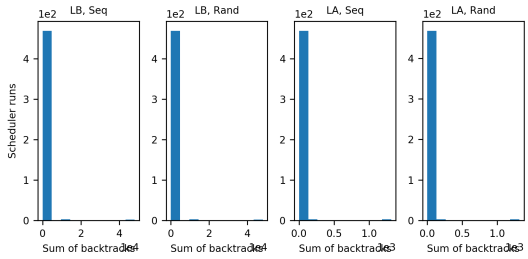
(c) *Boxplot with sum of backtracks per generated schedule.*



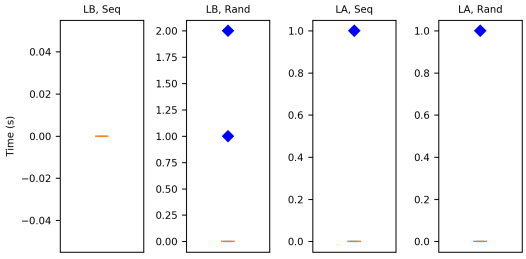
(d) *Boxplot with sum of backtracks until deem unschedulable.*



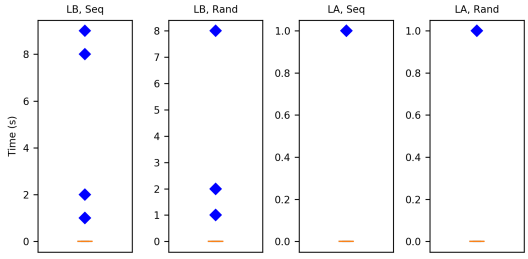
(e) *Histogram with sum of backtracks per generated schedule.*



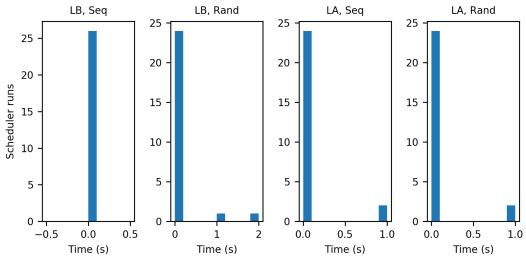
(f) *Histogram with sum of backtracks until deem unschedulable.*



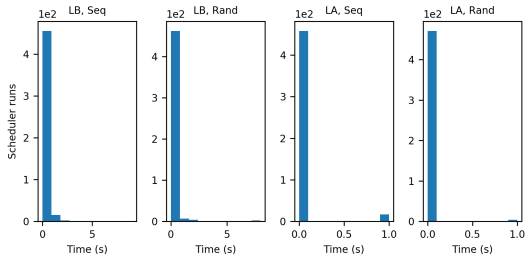
(g) *Boxplot with time to generate a schedule.*



(h) *Boxplot with time to deem unschedulable.*

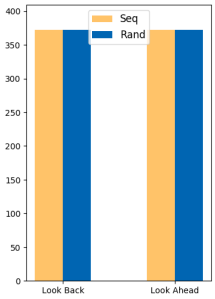


(i) *Histogram with time to generate a schedule.*

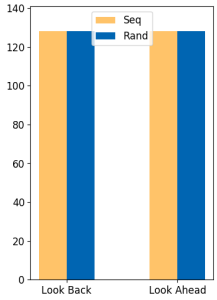


(j) *Histogram with time to deem unschedulable.*

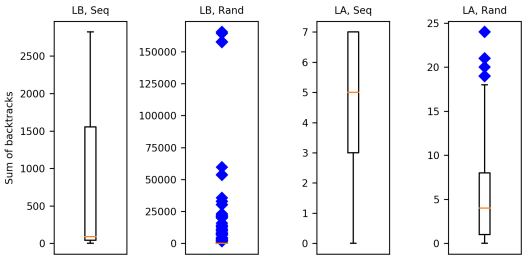
Figure V.18: *Experiment Cra1u9: STSTTN runs for 500 sets of virtual links. Sets scheduled by STSTTN on the left and not scheduled on the right. Data classified by traversing and edge selection algorithms.*



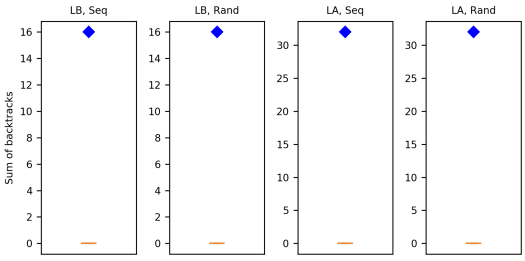
(a) *Number of scheduled sets.*



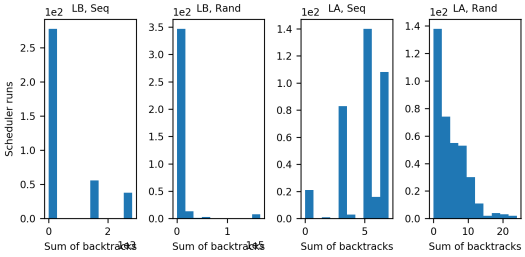
(b) *Number of not scheduled sets.*



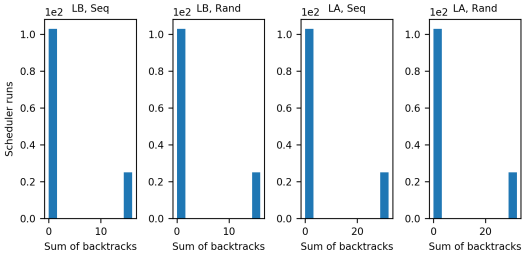
(c) *Boxplot with sum of backtracks per generated schedule.*



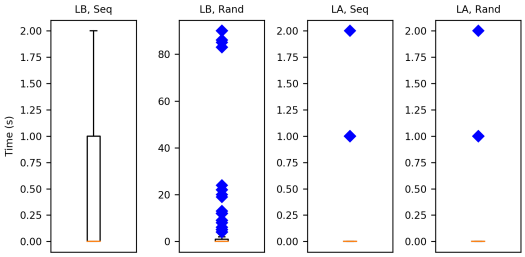
(d) *Boxplot with sum of backtracks until deem unschedulable.*



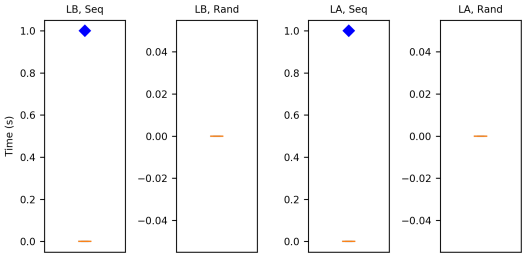
(e) *Histogram with sum of backtracks per generated schedule.*



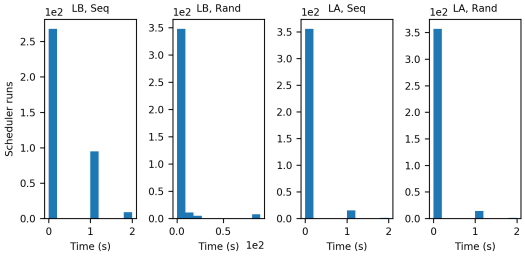
(f) *Histogram with sum of backtracks until deem unschedulable.*



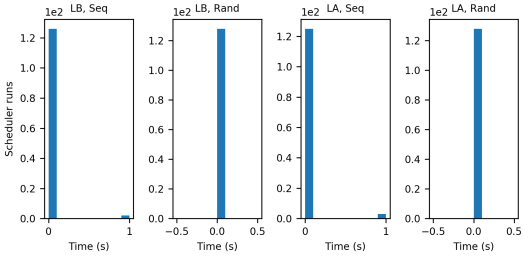
(g) *Boxplot with time to generate a schedule.*



(h) *Boxplot with time to deem unschedulable.*

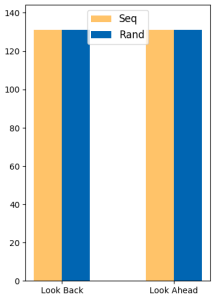


(i) *Histogram with time to generate a schedule.*

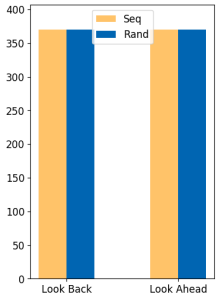


(j) *Histogram with time to deem unschedulable.*

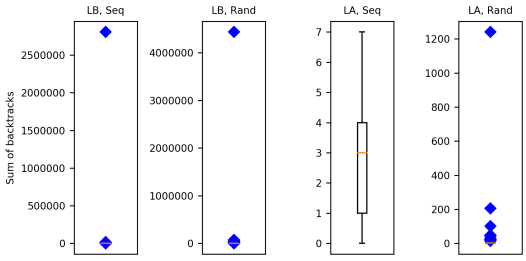
Figure V.19: *Experiment Cra2u5: STSTTN runs for 500 sets of virtual links. Sets scheduled by STSTTN on the left and not scheduled on the right. Data classified by traversing and edge selection algorithms.*



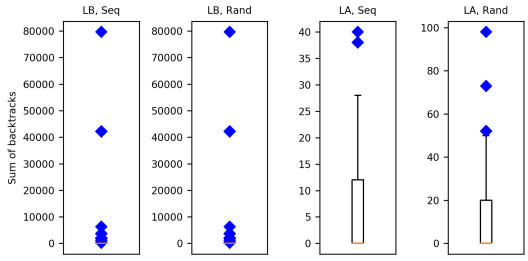
(a) *Number of scheduled sets.*



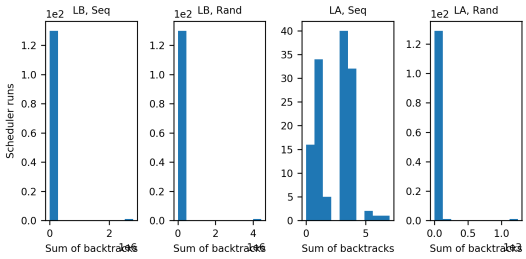
(b) *Number of not scheduled sets.*



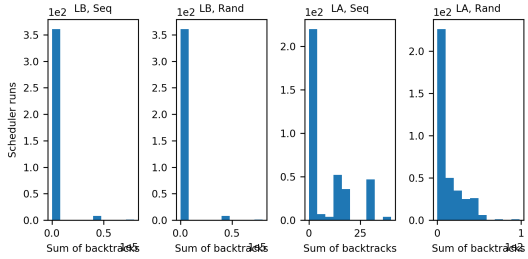
(c) *Boxplot with sum of backtracks per generated schedule.*



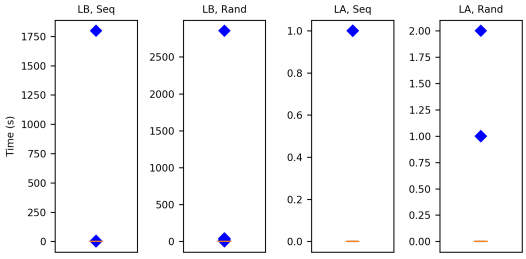
(d) *Boxplot with sum of backtracks until deem unschedulable.*



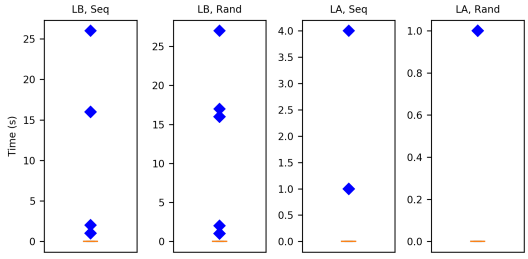
(e) *Histogram with sum of backtracks per generated schedule.*



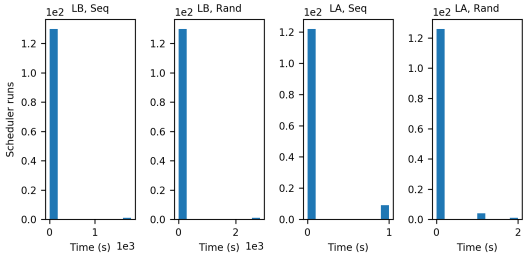
(f) *Histogram with sum of backtracks until deem unschedulable.*



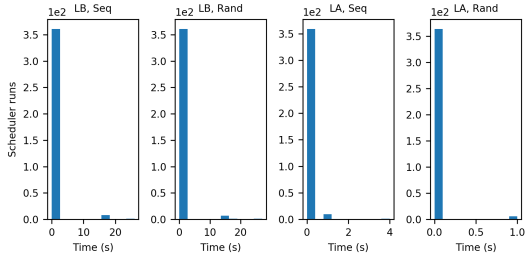
(g) *Boxplot with time to generate a schedule.*



(h) *Boxplot with time to deem unschedulable.*

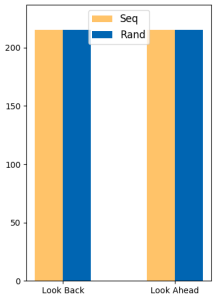


(i) *Histogram with time to generate a schedule.*

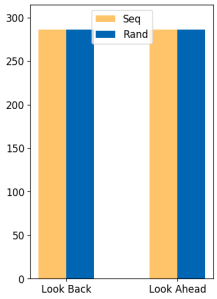


(j) *Histogram with time to deem unschedulable.*

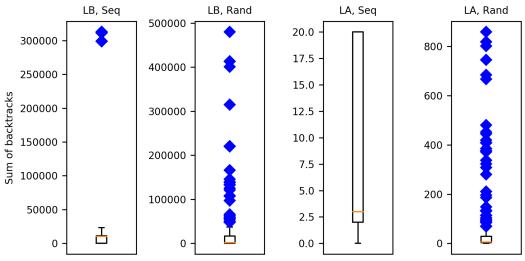
Figure V.20: *Experiment Cra2u7: STSTTN runs for 500 sets of virtual links. Sets scheduled by STSTTN on the left and not scheduled on the right. Data classified by traversing and edge selection algorithms.*



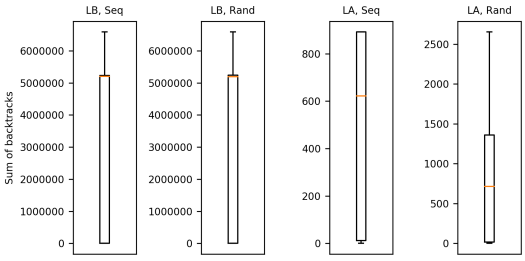
(a) *Number of scheduled sets.*



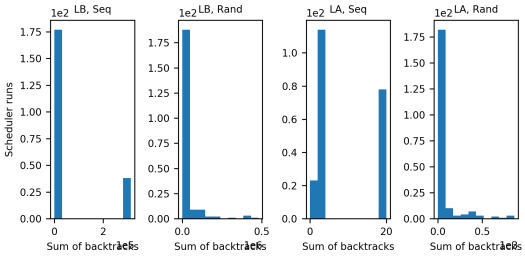
(b) *Number of not scheduled sets.*



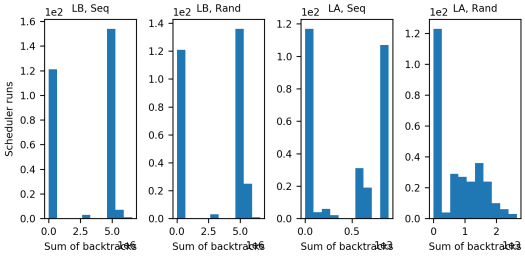
(c) *Boxplot with sum of backtracks per generated schedule.*



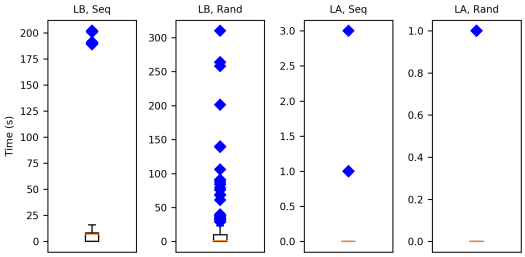
(d) *Boxplot with sum of backtracks until deem unschedulable.*



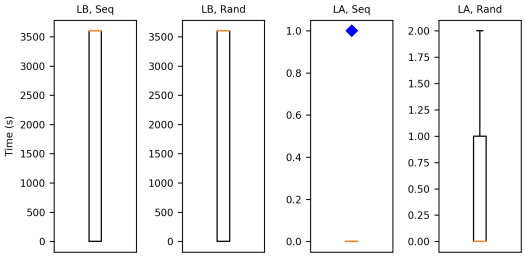
(e) *Histogram with sum of backtracks per generated schedule.*



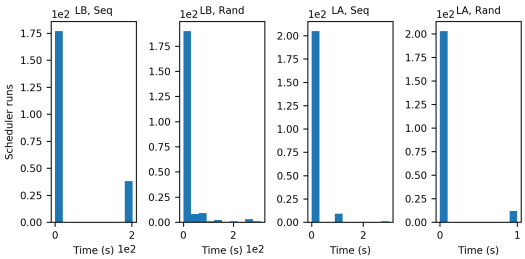
(f) *Histogram with sum of backtracks until deem unschedulable.*



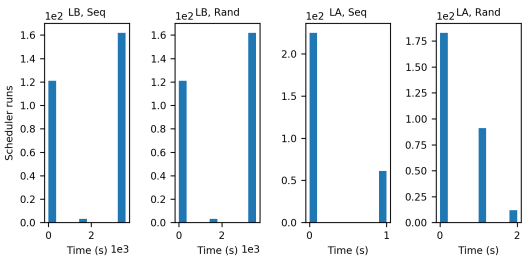
(g) *Boxplot with time to generate a schedule.*



(h) *Boxplot with time to deem unschedulable.*

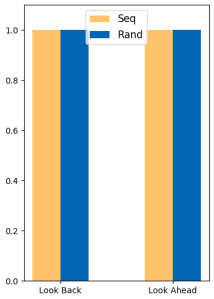


(i) *Histogram with time to generate a schedule.*

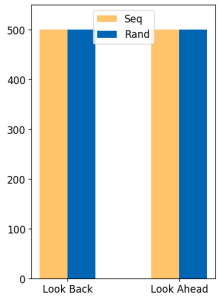


(j) *Histogram with time to deem unschedulable.*

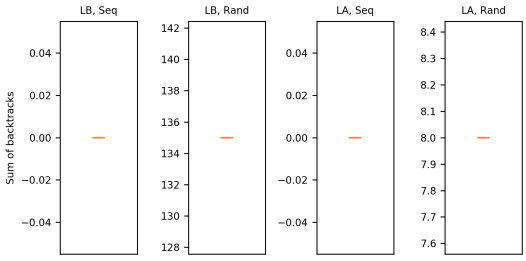
Figure V.21: *Experiment Cra2u8: STSTTN runs for 500 sets of virtual links. Sets scheduled by STSTTN on the left and not scheduled on the right. Data classified by traversing and edge selection algorithms.*



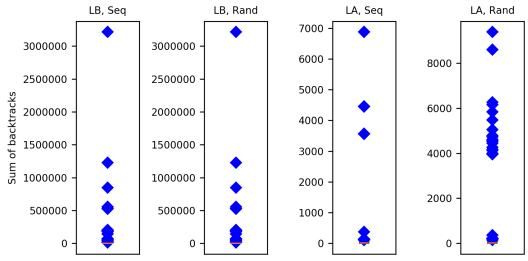
(a) *Number of scheduled sets.*



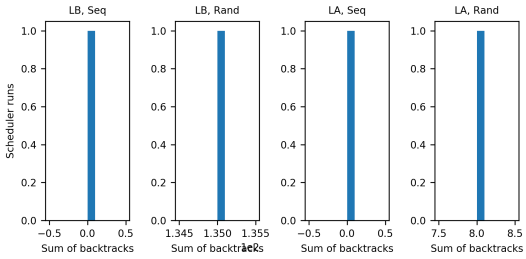
(b) *Number of not scheduled sets.*



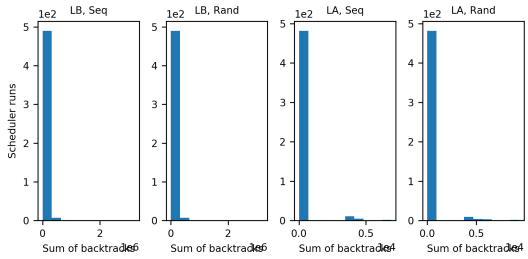
(c) *Boxplot with sum of backtracks per generated schedule.*



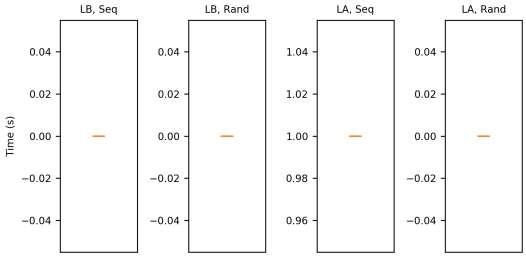
(d) *Boxplot with sum of backtracks until deem unschedulable.*



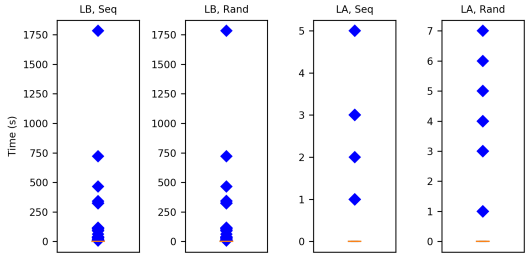
(e) *Histogram with sum of backtracks per generated schedule.*



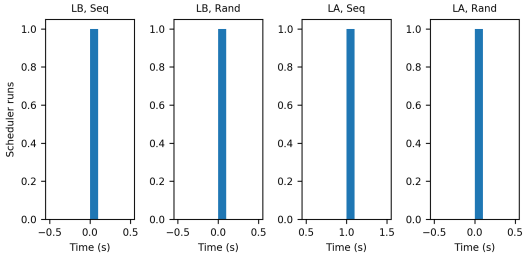
(f) *Histogram with sum of backtracks until deem unschedulable.*



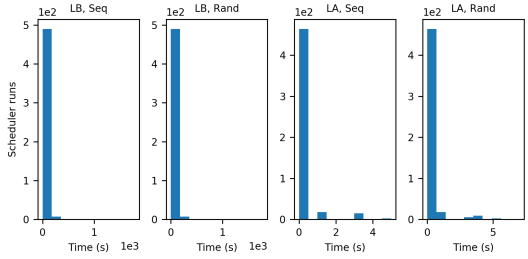
(g) *Boxplot with time to generate a schedule.*



(h) *Boxplot with time to deem unschedulable.*

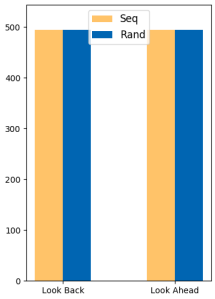


(i) *Histogram with time to generate a schedule.*

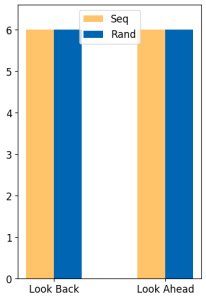


(j) *Histogram with time to deem unschedulable.*

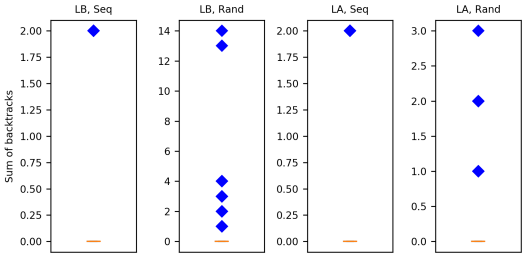
Figure V.22: *Experiment Cra2u9: STSTTN runs for 500 sets of virtual links. Sets scheduled by STSTTN on the left and not scheduled on the right. Data classified by traversing and edge selection algorithms.*



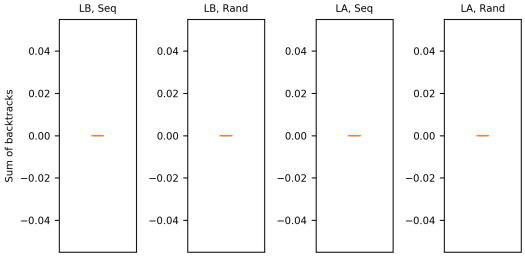
(a) *Number of scheduled sets.*



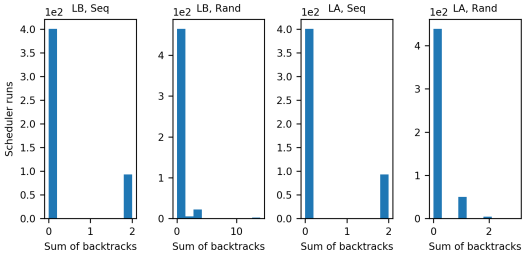
(b) *Number of not scheduled sets.*



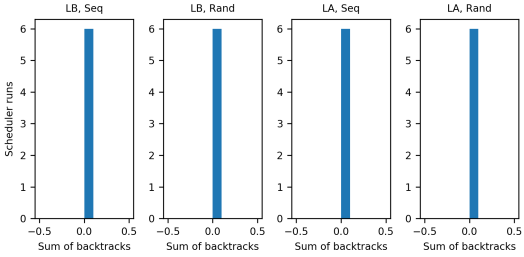
(c) *Boxplot with sum of backtracks per generated schedule.*



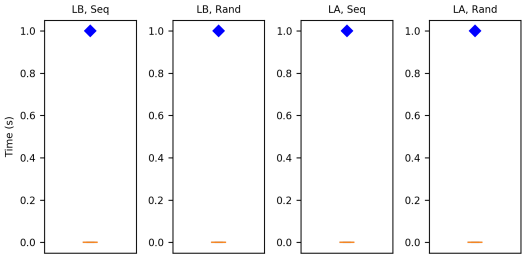
(d) *Boxplot with sum of backtracks until deem unschedulable.*



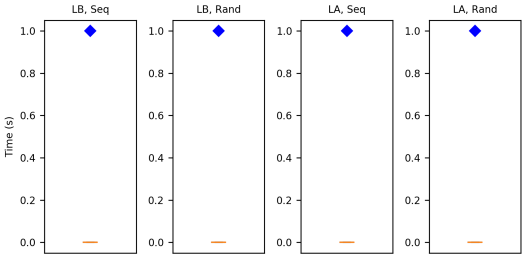
(e) *Histogram with sum of backtracks per generated schedule.*



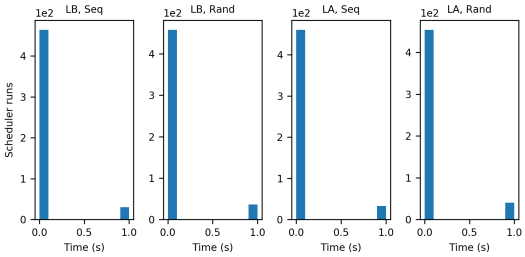
(f) *Histogram with sum of backtracks until deem unschedulable.*



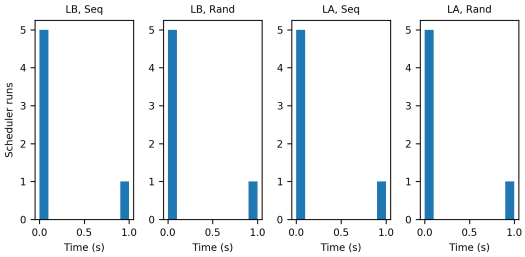
(g) *Boxplot with time to generate a schedule.*



(h) *Boxplot with time to deem unschedulable.*

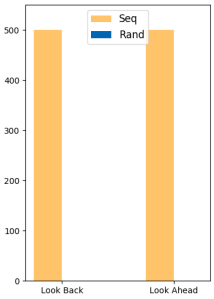


(i) *Histogram with time to generate a schedule.*

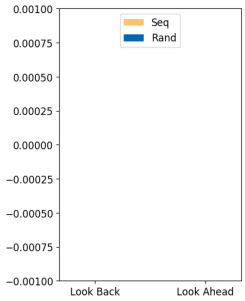


(j) *Histogram with time to deem unschedulable.*

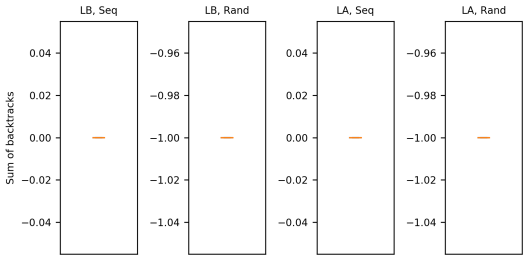
Figure V.23: *Experiment Cra3u5: STSTTN runs for 500 sets of virtual links. Sets scheduled by STSTTN on the left and not scheduled on the right. Data classified by traversing and edge selection algorithms.*



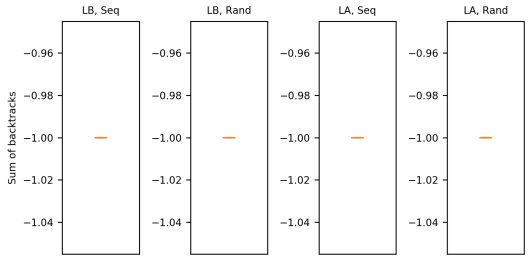
(a) *Number of scheduled sets.*



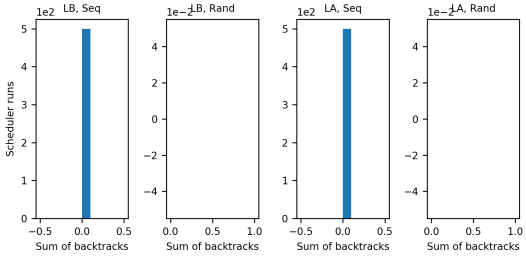
(b) *Number of not scheduled sets.*



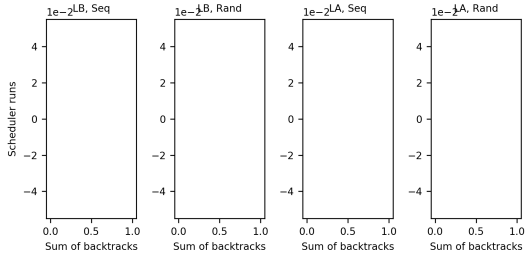
(c) *Boxplot with sum of backtracks per generated schedule.*



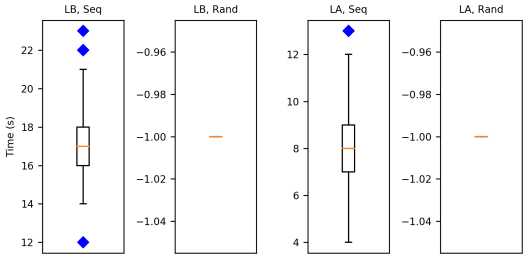
(d) *Boxplot with sum of backtracks until deem unschedulable.*



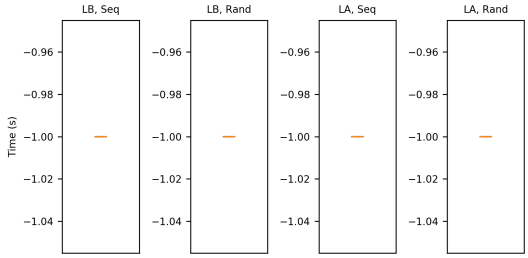
(e) *Histogram with sum of backtracks per generated schedule.*



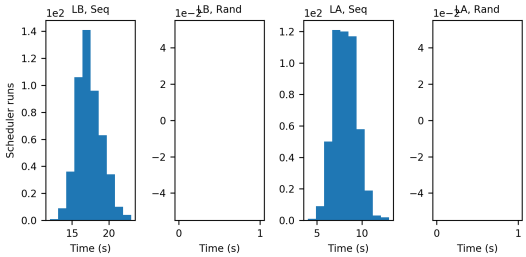
(f) *Histogram with sum of backtracks until deem unschedulable.*



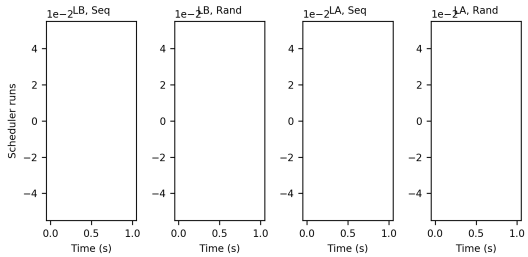
(g) *Boxplot with time to generate a schedule.*



(h) *Boxplot with time to deem unschedulable.*

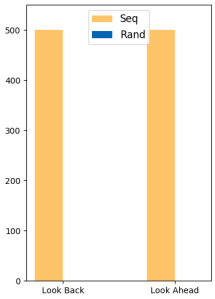


(i) *Histogram with time to generate a schedule.*

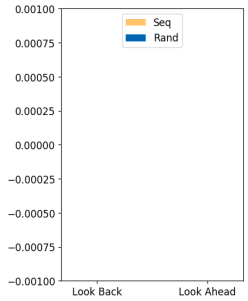


(j) *Histogram with time to deem unschedulable.*

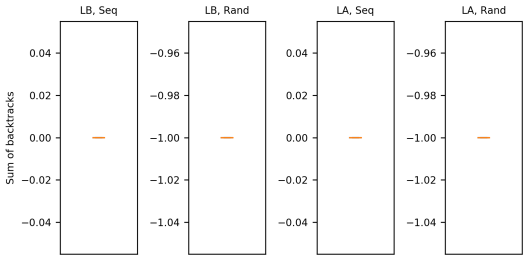
Figure V.24: *Experiment afdx1u8v200c1: STSTTN runs for 500 sets of virtual links. Sets scheduled by STSTTN on the left and not scheduled on the right. Data classified by traversing and edge selection algorithms.*



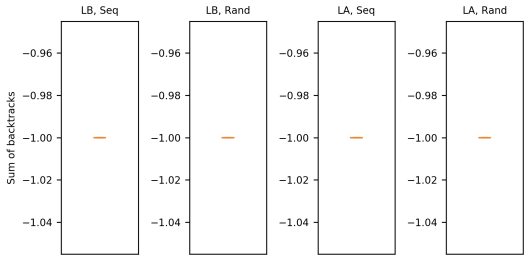
(a) *Number of scheduled sets.*



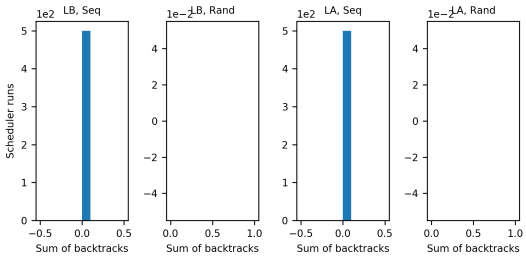
(b) *Number of not scheduled sets.*



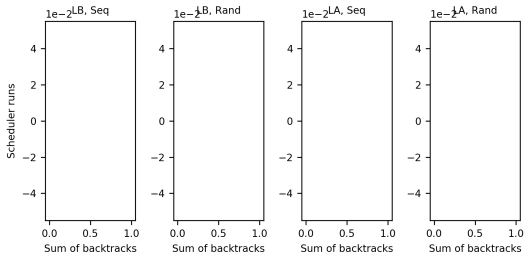
(c) *Boxplot with sum of backtracks per generated schedule.*



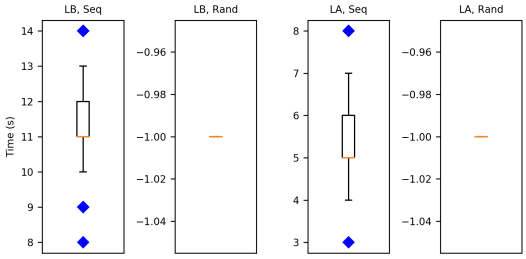
(d) *Boxplot with sum of backtracks until deem unschedulable.*



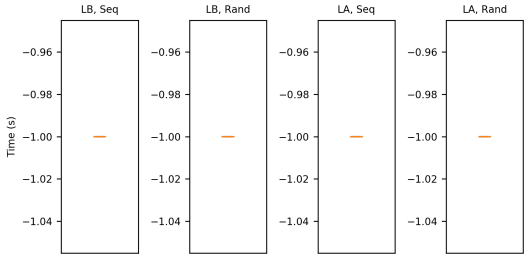
(e) *Histogram with sum of backtracks per generated schedule.*



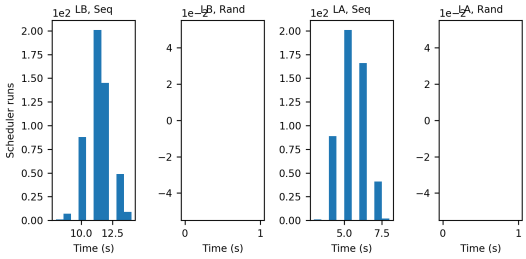
(f) *Histogram with sum of backtracks until deem unschedulable.*



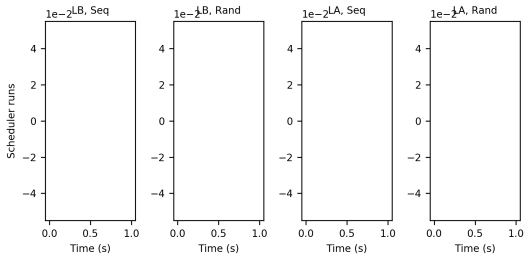
(g) *Boxplot with time to generate a schedule.*



(h) *Boxplot with time to deem unschedulable.*

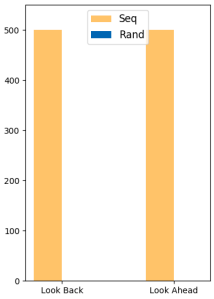


(i) *Histogram with time to generate a schedule.*

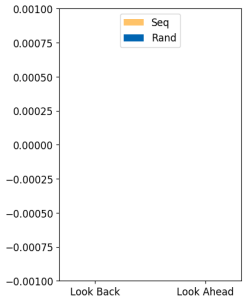


(j) *Histogram with time to deem unschedulable.*

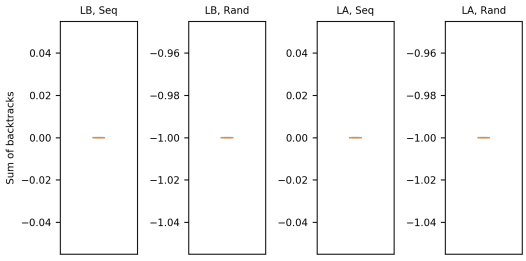
Figure V.25: *Experiment afdx1u9v200c1: STSTTN runs for 500 sets of virtual links. Sets scheduled by STSTTN on the left and not scheduled on the right. Data classified by traversing and edge selection algorithms.*



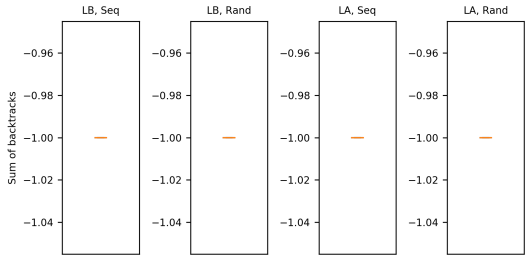
(a) *Number of scheduled sets.*



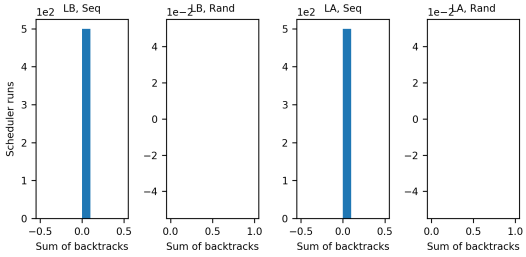
(b) *Number of not scheduled sets.*



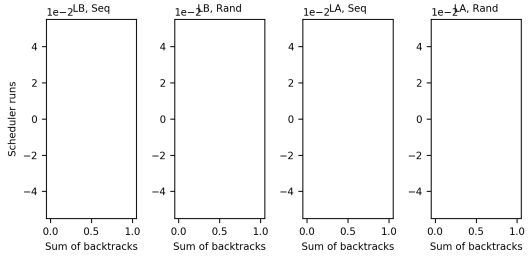
(c) *Boxplot with sum of backtracks per generated schedule.*



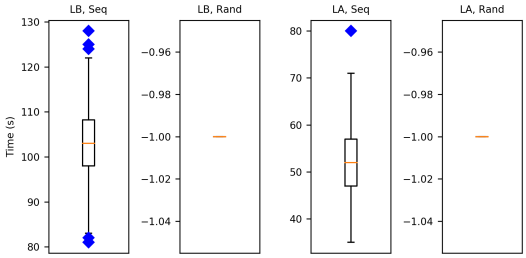
(d) *Boxplot with sum of backtracks until deem unschedulable.*



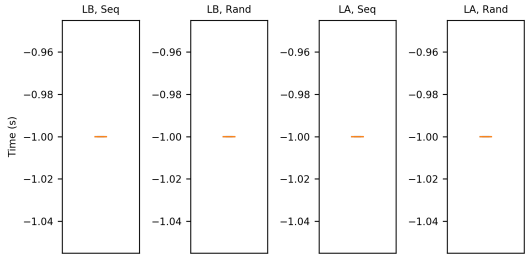
(e) *Histogram with sum of backtracks per generated schedule.*



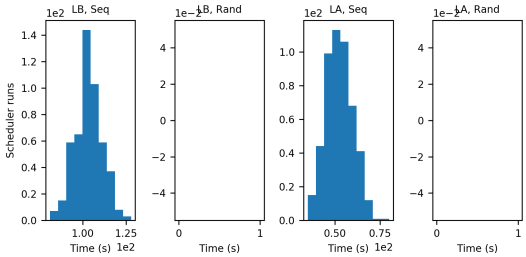
(f) *Histogram with sum of backtracks until deem unschedulable.*



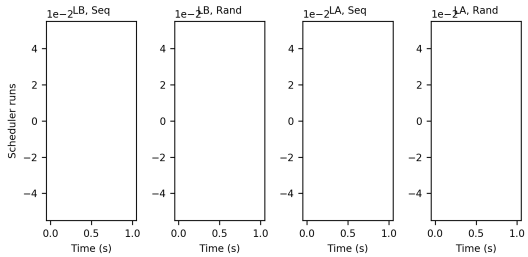
(g) *Boxplot with time to generate a schedule.*



(h) *Boxplot with time to deem unschedulable.*



(i) *Histogram with time to generate a schedule.*



(j) *Histogram with time to deem unschedulable.*

Figure V.26: *Experiment afdx1u9v400c1: STSTTN runs for 500 sets of virtual links. Sets scheduled by STSTTN on the left and not scheduled on the right. Data classified by traversing and edge selection algorithms.*

V.6 Summary

In this chapter, we proposed an off-line scheduler for Time-Triggered Ethernet called Search Tree based Scheduler for Time-Triggered Networks (STSTTN). The goal of STSTTN is to generate a schedule for the transmission of TT-frames in each physical link in the network, such that the time windows reserved for the transmission of TT-frames do not overlap in any physical link. We called *transmission window*, the timespan reserved for the transmission of a TT-frame. A *phase* represents the start time of the transmission window used to transmit the first frame of a virtual link.

We modeled this scheduling problem as a *search tree*, in which each virtual link is assigned to a level of the search tree. Each *vertex* in a level of the search tree represents 4 set with phases that can be used to schedule the virtual link on that level. Each *edge* represents a phase. The *path* leading to each vertex describes a schedule. STSTTN traverses the search tree until it reaches a leaf. If it reaches a valid leaf, the schedule for all virtual links traversing a physical link is found. If, on the contrary, it reaches an invalid leaf, the scheduler backtracks. If no further backtracking is possible, the set of virtual links is deemed unschedulable.

We presented two approaches to decrease the search space of the search tree called *look back* and *look ahead*. Both approaches compare, pairwise, the properties of virtual links scheduled along the visited path and compute a set of possibly feasible phases for the next virtual link. The difference between *look back* and *look ahead* is that the latter, additionally checks for the properties of virtual links which have not been scheduled yet. We presented an example showing that, even in a small set of virtual links, *look back* and *look ahead* can decrease the search space by one order of magnitude (14 vertices in Figure V.5 instead of 170 vertices in Figure V.3). We demonstrated that other criteria, e.g., the algorithms to assign virtual links to levels, and to select the edges, also impact the search space.

We showed that selecting a "bad" edge at upper levels of the tree might lead to infeasible schedules which can only be detected at deeper levels, leading to a large number of backtracks before the scheduler deems the vertex invalid. In order to allow for STSTTN to earlier discard such "bad" vertices, we presented a preventive tree pruning heuristic.

Traversing the search tree and backtracking imply different computational costs depending on the approach used to construct the search tree. For the *look back* approach, traversing and backtracking computational costs increase at deeper levels of the search tree. The opposite is true for the *look ahead* approach: traversing and backtracking computational costs decrease at deeper levels of the search tree.

The schedule of a virtual link, requires the selection of a phase in each traversed physical link. We showed that, in order to achieve the minimum transmission latency, a transmission window must start immediately after the frame is ready for transmission. We presented an approach to schedule the TT virtual links which allows for the minimum transmission latency.

Our experiments confirmed our hypotheses showing that: *Look ahead* is more efficient than *look back*; for a fixed set of periods, the higher the utilization of the virtual link set,

the harder it is for STSTTN to find a feasible schedule; the algorithms used to assign virtual links to levels of the search tree impacts on the search space; and that STSTTN can rapidly compute a feasible schedule for a set of virtual links with harmonic periods.

Chapter VI

Conclusions

For many years, most distributed real-time systems employed data communication systems specially tailored to address the specific requirements of individual domains: for instance, Controlled Area Network (CAN) and Flexray in the automotive domain, ARINC 429 [FW10] and TTP [Kop95] in the aerospace domain. Recently, the number of distributed real-time systems, as well as the number of nodes per system, have drastically increased. Further, new types of applications have been implemented on distributed systems, requiring large network bandwidth, e.g., entertainment video streams and backup cameras.

Ethernet (IEEE 802.3) [iee12] is a well established network standard. Further, it is fast, easy to install, and the interface ICs are cheap [Dec05]. However, Ethernet does not offer any temporal guarantee. Recently a number of protocols merging the benefits of Ethernet and the temporal guarantees required by distributed real-time systems have been proposed recently. In the aerospace domain, two of these solutions are: Avionics Full-Duplex Switched Ethernet (AFDX) [AFD09] and Time-Triggered Ethernet (TTEthernet) [tim16].

AFDX is a rate-constrained network based on switched Ethernet, implemented in most recent aircraft. Each AFDX communication channel defines a *virtual link*. The properties of the frames of a virtual link are defined off-line. Each virtual link defines: i) a static route, ii) a priority level, iii) a maximum frame size (L_{max}), iv) a minimum time between two consecutive frames (BAG), and v) a maximum jitter.

The unsynchronized message transmission of AFDX networks leads to data contention on the output ports of the switches. AFDX addresses this data contention by means of buffering. In order to avoid data loss caused by overflow of buffers on the output ports of AFDX switches, a network designer should carefully specify the amount of memory reserved for each output port buffer. Therefore, he/ she must compute an upper bound for the backlog of each output buffer in the network. Note that, due to the safety critical properties of avionics systems, buffer overflow and the resulting data loss must be avoided at all cost. Yet, large overprovision of memory to buffers should be avoided.

Computing the exact largest buffer backlog value of each buffer in an industrial size AFDX network is intractable: current methods to compute the exact largest buffer backlog relies on checking all possible combinations of ingress and egress times of frames. Considering the unpredicted dispatch time of AFDX frames, the computational require-

ments of those methods explode as the network traffic increases.

Although the current AFDX standard allows for traffic with only two priority levels, some AFDX commercial solutions allow for multiple priority traffic. The state-of-the-art AFDX buffer backlog analysis does not provide a method to compute deterministic upper bounds for buffer backlog of AFDX networks with multiple priority traffic.

TTEthernet is a real-time network based on switched Ethernet which allows for best-effort, rate-constrained and time-triggered traffic to coexist in the same communication system. TTEthernet provides a fault-tolerant transparent clock synchronization protocol, required to synchronize the nodes participating in the time-triggered message communication. In time-triggered networks, the transmission time of each frame is known to the sender and receiver(s) before run-time. A scheduling table stores the transmission and reception time window for each frame. This table is computed off-line and ensures that transmission windows (on a physical link) do not overlap.

The transmission (dispatch) time of each frame is defined by two parameters: *period* and *phase*, where *period* represents the constant timespan between the transmission of any two consecutive frames of the same message, and *phase* represents the amount of time between the start of the schedule until the point in time when the first frame of a message is scheduled.

VI.1 Overview of Contributions

We proposed in this dissertation, solutions for two problems faced during the design of AFDX and TTEthernet networks, as described in the following sections.

VI.1.1 Computation of Buffer Backlog Upper Bounds for AFDX Networks with Multiple Priorities

In this dissertation, we presented a method to compute an upper bound for the backlog on each output buffer of AFDX networks with multiple priority traffic. Our method is based on the largest busy period encountered by a frame of each virtual link that egresses a switch through the output port in which the buffer under analysis belongs to. Knowing which are the frames in the largest busy period, however, is not enough to determine the largest buffer backlog. We showed that the buffer backlog depends not only on the static properties (amount, size and priority) of the of the frames in the busy period, but also the order and the point in time in which they ingress and egress the buffers.

In order to simplify the analysis of the buffer backlog of AFDX networks with multiple priorities, we introduced the concept of *type of interval*. The idea behind these intervals is to analyze the ingress and egress of frames in the largest busy period, and identify intervals of time with the same properties of one of the *types of interval*. We showed that, according to the defined properties, only four types of intervals exist. Further, we presented the buffer backlog accrual for each type of interval.

We used the four types of interval to describe the properties of the scenario (ingress and egress order of frames in the largest busy period) leading to the largest backlog

encountered by a frame of a virtual link. We called this scenario *worst case scenario*. Once the worst case scenario is identified, we compute an upper bound for the buffer backlog encountered by a frame of *one* and then of *all* virtual links that egresses a switch through the output port in which the buffer under analysis belongs to. This computation provides an upper bound for the backlog of the buffer under analysis.

We further discussed how the results achieved by our method can be applied to AFDX switches that do not conform with our assumptions on how the ingress and egress of frames are handled by AFDX switches.

Our experiments showed that the computational cost of our method is negligible when compared to the cost of computing the end-to-end delay using the trajectory approach.

VI.1.2 A Strictly Periodic Scheduler for Time-Triggered Ethernet

We proposed in this dissertation, an off-line scheduler for time-triggered networks, in particular for TTEthernet, called Search Tree based Scheduler for Time-Triggered Networks (STSTTN). Our method formulates the scheduling problem as a set of search trees, each of them representing one physical link. *Vertices* in a level of the search tree represent the sets with phases that can be used to schedule the virtual link on that level, *Edges* represent the phases selected by the scheduler, and a *path* leading a vertex describes a schedule.

We presented two approaches to decrease the search space while traversing the search tree, called *look back* and *look ahead*. Both approaches compare, pairwise, the properties of virtual links scheduled along the visited path and compute a set of possibly feasible phases for the next virtual link. The difference between these two approaches is that *look ahead* additionally checks for the properties of virtual links which have not been scheduled yet.

STSTTN traverses the search tree until it reaches a leaf. If it reaches a valid leaf, the schedule for all virtual links traversing a physical link is found. If, on the contrary, it reaches an invalid leaf, the scheduler backtracks. If no further backtracking is possible, the set of virtual links is deemed unschedulable.

We further described how STSTTN allows to guide the selection of edges, and how to assign virtual links to tree levels. Additionally, we presented an optional pruning tree heuristic to reduce the search space.

Finally, in order to schedule the virtual links with minimum latency, we proposed to create one search tree for each physical link, where each search tree contains the virtual links that cross the corresponding physical link. We showed how to account for the minimum latency while computing the sets of phases represented by the vertices.

Our experiments showed that *look ahead* is more efficient than *look back*. Further, they showed that for a fixed set of periods, the higher the utilization of the virtual link set, the harder it is for STSTTN to find a feasible schedule. We could also observe that, the algorithms used to assign virtual links to levels of the search tree impacts on the search space, and that STSTTN can rapidly compute a feasible schedule for a set of virtual links with harmonic periods.

VI.2 Future Work

The contributions presented in this dissertation also show opportunities for future research. One possible extension to the buffer backlog upper bound computation method presented in this dissertation is to provide an upper bound on the number of frames, instead of on the amount of memory, stored in the output buffers of AFDX networks with multiple priority traffic, Similar analysis has been propose by Benammar in [BBRR], however solely for single priority traffic. Another possible future work is addressing the different switch designs during the analysis of the worst case scenario, instead of adding the pessimism at the end of the upper bound computations, We believe that this approach would lead to less pessimism. A third possible research opportunity is to propose an alternative method to compute the *competing frames*, thus reducing the intrinsic pessimism on the computation of the busy period introduced by methods like the trajectory approach.

Our proposed search tree based schedule for time-triggered networks (STSTTN) also provided some pointers for future research. A natural possible extension to STSTTN is to relax the assumption that the schedule must provide the minimum traversal time. For many applications, the traversal time need not to be minimum and the need for buffering of TT-frames is acceptable. Certainly, providing auto tuning capabilities to STSTTN is a challenging possible extension. For instance, depending on the properties of the set of virtual links to be scheduled, STSTTN can decide on the most effective algorithms for assigning virtual links to search tree levels, select edges, etc.

Appendix A

Proof of Theorem III.1

This appendix proves that the value computed by Equation (III.9) is larger than or equal to zero for any value of τ such that $\omega^{\lambda} \leq \tau \leq \theta^{\lambda P}$.

The amount of transmitted P-data from τ until θ^{*P} in the worst case scenario, is represented by $T^{*P}_{(\theta^{*P}-\tau)}$ and computed by Equation (A.1).

$$T^{*P}_{(\theta^{*P}-\tau)} = \Delta^{*P} - T^{*P}_{\tau} \tag{A.1}$$

In the worst case scenario, no P-data is transmitted before β^*. Thus, for $\tau \geq \beta^*$, the amount of data transmitted until τ is equal to $(\tau - \beta^*)$; otherwise it is equal to zero. Therefore:

$$T^{*P}_{\tau} = d(\tau - \beta^*)^+ \tag{A.2}$$

And, consequently:

$$T^{*P}_{(\theta^{*P}-\tau)} = \Delta^{*P} - d(\tau - \beta^*)^+$$

Let $T^{\lambda P}_{\theta^{\lambda P}}$ be the total amount of P-data transmitted until $\theta^{\lambda P}$ in scenario λ. Considering that the worst case scenario is the scenario in which less P-data is transmitted until the transmission of the latest P-frame, we have:

$$\Delta^{*P} = T^{\lambda P}_{\theta^{\lambda P}} - K \mid K \geq 0$$

Thus:

$$T^{*P}_{(\theta^{*P}-\tau)} = T^{\lambda P}_{\theta^{\lambda P}} - K - d(\tau - \beta^*)^+ \tag{A.3}$$

After combining with Equations (III.4), (A.2) and (A.3), Equation (III.9) becomes:

$$B^{*P}_{\theta^{*P}} - B^{\lambda P}_{\tau} = R^{\lambda P}_{(\theta^{\lambda P}-\tau)} - T^{\lambda P}_{\theta^{\lambda P}} + K + d(\tau - \beta^*)^+ + T^{\lambda L}_{\tau} - d(\tau - \beta^*)^+$$

Considering:

$$T^{\lambda P}_{(\theta^{\lambda P}-\tau)} = T^{\lambda P}_{\theta^{\lambda P}} - T^{\lambda L}_{\tau}$$

then:

$$B^{*P}_{\theta^{*P}} - B^{\lambda P}_{\tau} = R^{\lambda P}_{(\theta^{\lambda P}-\tau)} - T^{\lambda P}_{(\theta^{\lambda P}-\tau)} + K \tag{A.4}$$

We present the analysis of Equation (A.4) for two types of scenarios: first, for scenarios in which $\beta^{\lambda} > \theta^{\lambda P}$, then, for those scenarios in which $\beta^{\lambda} \leq \theta^{\lambda P}$.

A.1 Scenarios with $\beta^{\curlywedge} > \theta^{\curlywedge P}$

According to Section III.2.6.1, the worst case scenario requires the largest β^* and the shortest θ^{*P}. Hence $\beta^{\curlywedge} \leq \beta^*$ and $\theta^{*P} \geq \theta^{*P}$.

If, in the worst case scenario, $\beta^* < \theta^{*P}$, then $\beta^{\curlywedge} < \theta^{\curlywedge P}$. Consequently, there exists no scenario $\curlywedge$ in which $\beta^{\curlywedge} > \theta^{\curlywedge P}$ for $\beta^* < \theta^{*P}$.

Otherwise, if in the worst case scenario, $\beta^* \geq \theta^{*P}$, i.e., $\Delta^{*P} \leq 0$, then the buffer backlog upper bound is equal to the total amount of P-data that ingress this buffer, i.e, σ^P_{ALL} (see Equation III.18). Logically, there exists no scenario $\curlywedge$ in which the buffer backlog is larger than σ^P_{ALL}. ■

A.2 Scenarios with $\beta^{\curlywedge} \leq \theta^{\curlywedge P}$

We divide this analysis into two cases, according to the location of τ: $\beta^{\curlywedge} \leq \tau \leq \theta^{\curlywedge P}$ and $\omega^{\curlywedge} \leq \tau < \beta^{\curlywedge}$.

A.2.1 $\beta^{\curlywedge} \leq \tau \leq \theta^{\curlywedge P}$

Assuming no idle time on the input links, and considering that the last frame with priority other than P arrives at $\beta^{\curlywedge}$ in scenario $\curlywedge$, the interval $[\tau, \theta^{\curlywedge P}]$ is of type 2. Per definition, the amount of ingress P-data is larger or equal than the egress P-data in this interval (see Section III.2.4) and therefore $R^{\curlywedge P}_{(\theta^{\curlywedge P} - \tau)} \geq T^{\curlywedge P}_{(\theta^{\curlywedge P} - \tau)}$. Thus Equation (A.4) and consequently Equation (III.9) result in positive values for $\beta^{\curlywedge} \leq \tau \leq \theta^{\curlywedge P}$. ■

A.2.2 $\omega^{\curlywedge} \leq \tau < \beta^{\curlywedge}$

Let us analyze the terms $R^{\curlywedge P}_{(\theta^{\curlywedge P} - \tau)}$ and $T^{\curlywedge P}_{(\theta^{\curlywedge P} - \tau)}$ of Equation (A.4):

$$\begin{aligned} R^{\curlywedge P}_{(\theta^{\curlywedge P} - \tau)} &= R^{\curlywedge P}_{(\theta^{\curlywedge P} - \beta^{\curlywedge})} + R^{\curlywedge P}_{(\beta^{\curlywedge} - \tau)} \\ T^{\curlywedge P}_{(\theta^{\curlywedge P} - \tau)} &= T^{\curlywedge P}_{(\theta^{\curlywedge P} - \beta^{\curlywedge})} + T^{\curlywedge P}_{(\beta^{\curlywedge} - \tau)} \end{aligned}$$

thus:

$$R^{\curlywedge P}_{(\theta^{\curlywedge P} - \tau)} - T^{\curlywedge P}_{(\theta^{\curlywedge P} - \tau)} = R^{\curlywedge P}_{(\theta^{\curlywedge P} - \beta^{\curlywedge})} - T^{\curlywedge P}_{(\theta^{\curlywedge P} - \beta^{\curlywedge})} + R^{\curlywedge P}_{(\beta^{\curlywedge} - \tau)} - T^{\curlywedge P}_{(\beta^{\curlywedge} - \tau)}$$

Notice that, $[\beta^{\curlywedge}, \theta^{\curlywedge P}]$ is a type 2 interval and per definition $R^{\curlywedge P}_{(\theta^{\curlywedge P} - \beta^{\curlywedge})} - T^{\curlywedge P}_{(\theta^{\curlywedge P} - \beta^{\curlywedge})} \geq 0$.

To prove that $R^{\curlywedge P}_{(\beta^{\curlywedge} - \tau)} - T^{\curlywedge P}_{(\beta^{\curlywedge} - \tau)} \geq 0$, we analyze the interval $[\tau, \beta^{\curlywedge}[$. The transmission of all frames with priorities other than P, obviously including those that arrive in the interval $[\tau, \beta^{\curlywedge}[$, ends at $\beta^{\curlywedge}$. We can draw two conclusions from this observation: first, the amount of HP-data and LP-data received in this interval is less than or equal to the length of the interval; second, this amount of data is less than the sum of the amount of HP-data plus the amount of LP-data transmitted in this interval. We express these

two conclusions in the next inequalities:

$$\begin{aligned} R^{\curlywedge HP}_{(\beta^{\curlywedge}-\tau)} + R^{\curlywedge LP}_{(\beta^{\curlywedge}-\tau)} &< d(\beta^{\curlywedge}-\tau) \\ R^{\curlywedge HP}_{(\beta^{\curlywedge}-\tau)} + R^{\curlywedge LP}_{(\beta^{\curlywedge}-\tau)} &\leq T^{\curlywedge HP}_{(\beta^{\curlywedge}-\tau)} + T^{\curlywedge LP}_{(\beta^{\curlywedge}-\tau)} \end{aligned} \tag{A.5}$$

Due to possible arrival of P-frames from multiple virtual links, the amount of P-data arriving between τ and $\beta^{\curlywedge}$ is larger than the length of this interval minus the sum of the amount of HP-data and the amount of LP-data that ingress in this interval, i.e.:

$$R^{\curlywedge P}_{(\beta^{\curlywedge}-\tau)} \geq d(\beta^{\curlywedge}-\tau) - (R^{\curlywedge HP}_{(\beta^{\curlywedge}-\tau)} + R^{\curlywedge LP}_{(\beta^{\curlywedge}-\tau)}) \tag{A.6}$$

Since there is no idle time on the output link between τ and $\beta^{\curlywedge}$, the amount of transmitted P-data in this interval is:

$$T^{\curlywedge P}_{(\theta^{\curlywedge P}-\beta^{\curlywedge})} = d(\beta^{\curlywedge}-\tau) - (T^{\curlywedge HP}_{(\beta^{\curlywedge}-\tau)} + T^{\curlywedge LP}_{(\beta^{\curlywedge}-\tau)}) \tag{A.7}$$

Subtracting (A.7) from (A.6) we have:

$$R^{\curlywedge P}_{(\beta^{\curlywedge}-\tau)} - T^{\curlywedge P}_{(\beta^{\curlywedge}-\tau)} \geq (T^{\curlywedge HP}_{(\beta^{\curlywedge}-\tau)} + T^{\curlywedge LP}_{(\beta^{\curlywedge}-\tau)}) - (R^{\curlywedge HP}_{(\beta^{\curlywedge}-\tau)} + R^{\curlywedge LP}_{(\beta^{\curlywedge}-\tau)})$$

According to (A.5):

$$(T^{\curlywedge HP}_{(\beta^{\curlywedge}-\tau)} + T^{\curlywedge LP}_{(\beta^{\curlywedge}-\tau)}) - (R^{\curlywedge HP}_{(\beta^{\curlywedge}-\tau)} + R^{\curlywedge LP}_{(\beta^{\curlywedge}-\tau)}) \geq 0$$

thus:

$$R^{\curlywedge P}_{(\beta^{\curlywedge}-\tau)} - T^{\curlywedge P}_{(\beta^{\curlywedge}-\tau)} \geq 0$$

Consequently, Equation (A.4) and Equation (III.9) result in positive values for $\omega^{\curlywedge} \leq \tau < \beta^{\curlywedge}$. ■

Appendix B

Computing Competing Frames with Trajectory Approach

The trajectory approach computes the busy period on every node visited by the virtual link under analysis (v), and the maximum number of frames (competing frames) of each virtual link that delays the latest arriving frame (f^m) of the virtual link v. According to [BSF10], the trajectory approach uses Equation (B.1) to compute an upper bound for the end-to-end delay of a frame of any virtual link v:

$$R_v = \max_{t \geq 0}(W^o_{v,t} + C_v - t) \tag{B.1}$$

where: C_v is the transmission time of a frame of virtual link v and $W^o_{v,t}$ is an upper bound on the latest arrival time of a frame of VL v generated at time t on node o. Equation (B.2) presents the computation of $W^o_{v,t}$.

$$W^o_{v,t} = \tag{B.2}$$

$$\sum_{\substack{j \in sp_v \cup \{v\} \\ \mathcal{P}_j \cap \mathcal{P}_v \neq \emptyset}} \left(1 + \left\lfloor \frac{t + A_{v,j}}{T_j} \right\rfloor\right) \times C_j \tag{B.2.1}$$

$$+ \sum_{\substack{j \in hp_v \\ \mathcal{P}_j \cap \mathcal{P}_v \neq \emptyset}} \left(1 + \left\lfloor \frac{W^o_{v,t} + B_{v,j}}{T_j} \right\rfloor\right) \times C_j \tag{B.2.2}$$

$$+ \sum_{\substack{h \in \mathcal{P}_v \\ h \neq last_v}} \left(\max_{\substack{h \in hp_v \cup sp_v \cup [v] \\ h \in \mathcal{P}_j}} \{C_j\} \right) \tag{B.2.3}$$

$$+ (|\mathcal{P}_v| - 1) \times L_{max} \tag{B.2.4}$$

$$+ \sum_{h \in \mathcal{P}_v} \delta^h_v \tag{B.2.5}$$

$$+ \sum_{\substack{h \in \mathcal{P}_v \\ h \neq first_v}} \Delta_h \tag{B.2.6}$$

$$- C_v \tag{B.2.7}$$

Term (B.2.1) represents the delay accrual caused by all the competing frames with the same priority as the virtual link under analysis, from its source to its destination node. Term (B.2.2) is the counterpart of Term (B.2.1) for competing frames with priorities higher than the virtual link under analysis. $A_{v,j}$ and $B_{v,j}$ represent time windows that define the earliest and latest generation time of a frame of VL j, respectively of same or higher priority than VL v, that might delay f^m on node o; δ_v^h represents the size of the largest low priority competing frame on node o; Term (B.2.5) represents the blocking time (impact due to frames of priority lower than f^m). Terms (B.2.3), (B.2.4) and (B.2.7) represent the same properties as explained in Section II.3.2.2 Notice that, in order to be consistent with the notation used in Chapter III, the indices of Equations (B.1) and (B.2) are not the same as those used in [BSF12b].

Formula (B.3) computes the length of the competing LP-frame.

$$\sum_{\substack{j \in \mathcal{V}^{o,LP,\bar{i}} \\ \forall \bar{i} \in \mathcal{I}^o}} \max(f_j) \tag{B.3}$$

The competing frames of same and higher priority than the virtual link v can be extracted from terms (B.2.1) and (B.2.2), using formulas (B.4) and (B.5), respectively. Notice that the value $W_{v,t}^o$ used in the computation of Formula (B.5) is the solution of the respective fixed point iterated Equation (B.2).

$$\left(1 + \left\lfloor \frac{t + A_{v,j}}{T_j} \right\rfloor\right) \mid j \in \mathcal{V}^{o,SP,j}, \forall\ j \in \mathcal{I}^o \tag{B.4}$$

$$\left(1 + \left\lfloor \frac{W_{v,t}^o + B_{v,j}}{T_j} \right\rfloor\right) \mid j \in \mathcal{V}^{o,HP,j}, \forall\ j \in \mathcal{I}^o \tag{B.5}$$

It is important to notice that, the values obtained by the trajectory approach might be pessimistic. Nevertheless, the method presented in Chapter III can be used with any other approach which computes the competing frames in the largest busy period encountered by frames of each virtual link. For further details on the trajectory approach, we refer the interested reader to [MM06b] and [BSF10].

Bibliography

[AFD09] *Aircraft Data Network Part 7. Avionics Full-Duplex Switched Ethernet Network. ARINC Specification 664 P7-1.* September 2009

[ASEF10] Adnan, M. ; Scharbarg, J. L. ; Ermont, J. ; Fraboul, C.: Model for worst case delay analysis of an AFDX network using timed automata. In: *2010 IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, 2010, S. 1–4

[BB05] Bini, Enrico ; Buttazzo, Giorgio C.: Measuring the performance of schedulability tests. 30 (2005), Nr. 1, S. 129–154

[BBRR] Benammar, Nassima ; Bauer, Henri ; Ridouard, FrÃ©dÃ©ric ; Richard, Pascal: Tighter buffer dimensioning in AFDX networks. 13, Nr. 4, 37–42. http://dl.acm.org/citation.cfm?id=3015043

[BMN11] Boyer, Marc ; Migge, Jörn ; Navet, Nicolas: A simple and efficient class of functions to model arrival curve of packetised flows. In: *1st International Workshop on Worst-case Traversal Time, in conj. with the 32nd IEEE Real-Time Systems Symposium (RTSS 2011), Vienna*, 2011

[BSF09] Bauer, H. ; Scharbarg, J. L. ; Fraboul, C.: Applying and optimizing trajectory approach for performance evaluation of AFDX avionics network. In: *2009 IEEE Conference on Emerging Technologies Factory Automation*, 2009, S. 1–8

[BSF10] Bauer, Henri ; Scharbarg, Jean-Luc ; Fraboul, Christian: Improving the Worst-Case Delay Analysis of an AFDX Network Using an Optimized Trajectory Approach. 6 (2010), Nr. 4, 521–533. http://dx.doi.org/10.1109/TII.2010.2055877. – DOI 10.1109/TII.2010.2055877. – ISSN 1551–3203, 1941–0050

[BSF12a] Bauer, H. ; Scharbarg, J.-L. ; Fraboul, C.: Worst-Case Backlog Evaluation of Avionics Switched Ethernet Networks with the Trajectory Approach. In: *ECRTS*, 2012, S. 78 –87

[BSF12b] Bauer, Henri ; Scharbarg, Jean-Luc ; Fraboul, Christian: Applying Trajectory approach with static priority queuing for improving the use of available AFDX resources. 48 (2012), Nr. 1, 101–133. http://dx.doi.org/10.1007/s11241-011-9142-9. – DOI 10.1007/s11241–011–9142–9. – ISSN 0922–6443, 1573–1383

[BSF12c] Bauer, Henri ; Scharbarg, Jean-Luc ; Fraboul, Christian: Worst-Case Backlog Evaluation of Avionics Switched Ethernet Networks with the Trajectory Approach, IEEE, 2012. – ISBN 978–1–4673–2032–0, 78–87

[But05] Buttazzo, Giorgio C.: *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications.* 2nd Edition. Springer Science+Business Media, 2005

[But10] Butz, Henning: Open integrated modular avionic (ima): State of the art and future development road map at airbus deutschland. 10 (2010), 1000. http://www.academia.edu/download/39499480/ima-henning-butz.pdf

[CLRS01] Cormen, Thomas H. ; Leiserson, Charles E. ; Rivest, Ronald L. ; Stein, Clifford: *Introduction to Algorithms, Second Edition.* The MIT Press, 2001. – ISBN 0262531968

[CO16] Craciunas, Silviu S. ; Oliver, Ramon S.: Combined task- and network-level scheduling for distributed time-triggered systems. 52 (2016), Nr. 2, 161–200. http://dx.doi.org/10.1007/s11241-015-9244-x. – DOI 10.1007/s11241–015–9244–x. – ISSN 0922–6443, 1573–1383

[Cru91a] Cruz, Rene L.: A calculus for network delay. Part I. Network elements in isolation. 37 (1991), Nr. 1, 114–131. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=61109

[Cru91b] Cruz, Rene L.: A calculus for network delay. Part II: Network Analysis. 37 (1991), Nr. 1, 114–131. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=61109

[DBBL07] Davis, Robert ; Burns, Alan ; Bril, Reinder ; Lukkien, Johan: Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. In: *Real-Time Systems Journal* vol. 35 (2007), S. 239 – 272

[Dec05] Decotignie, J.-D.: Ethernet-Based Real-Time and Industrial Communications. 93 (2005), Nr. 6, 1102–1117. http://dx.doi.org/10.1109/JPROC.2005.849721. – DOI 10.1109/JPROC.2005.849721. – ISSN 0018–9219

[FFG06] Frances, Fabrice ; Fraboul, Christian ; Grieu, Jérôme: Using network calculus to optimize the AFDX network. (2006). http://oatao.univ-toulouse.fr/2159/

[Foh94] Fohler, Gerhard J.: *Flexibility in Statically Scheduled Real-Time Systems*. Wien, Österreich, Diss., April 1994. https://pdfs.semanticscholar.org/c3e0/2fd96e588b8a4a8c6b479f8bca66e9a1b0de.pdf

[FW10] Frodyma, Pat ; Waldmann, B: *ARINC 429 Specification Tutorial*. https://www.aim-online.com/pdf/OVIEW429.PDF. Version: 2010

[GM01] Goossens, Joel ; Macq, Christophe: Limitation of the Hyper-Period in Real-Time Periodic Task Set Generation. In: *In Proceedings of the RTS Embedded System (RTS'01)*, 2001, S. 133–147

[Goo03] Goossens, Joël: Scheduling of Offset Free Systems. In: *Real-Time Systems* 24 (2003), S. 239–258. http://dx.doi.org/10.1023/A:1021782503695. – DOI 10.1023/A:1021782503695. – ISSN 0922–6443

[GPGH14] Gutiérrez, J. J. ; Palencia, J. C. ; González Harbour, Michael: Holistic schedulability analysis for multipacket messages in AFDX networks. 50 (2014), Nr. 2, 230–269. http://dx.doi.org/10.1007/s11241-013-9192-2. – DOI 10.1007/s11241–013–9192–2. – ISSN 0922–6443, 1573–1383

[Gro17] Group, Time-Sensitve Networking T.: *IEEE 802.1 Time-Sensitve Networking Task Group*. http://www.ieee802.org/1/pages/tsn.html. Version: 2017

[HF17] Heilmann, Florian ; Fohler, Gerhard: Reducing the End-To-End Delay of Rate-Constrained Traffic in TTEthernet Networks. In: *15th Workshop on Real-Time Networks (RTN'17) in conjuction with 29th Euromicro International Conference on Real-time Systems (ECRTS'17)*, 2017

[iee12] IEEE Standard for Ethernet. In: *IEEE Std 802.3-2012 (Revision to IEEE Std 802.3-2008)* (2012), Dec, S. 1–3747. http://dx.doi.org/10.1109/IEEESTD.2012.6419735. – DOI 10.1109/IEEESTD.2012.6419735

[KALW91] Korst, Jan ; Aarts, Emile ; Lenstra, Jan K. ; Wessels, Jaap: Periodic multiprocessor scheduling. Version: 1991. https://doi.org/10.1007/BFb0035103. In: Aarts, Emile H. L. (Hrsg.) ; Leeuwen, Jan van (Hrsg.) ; Rem, Martin (Hrsg.): *PARLE '91 Parallel Architectures and Languages Europe: Volume I: Parallel Architectures and Algorithms Eindhoven, The Netherlands, June 10–3, 1991 Proceedings*. Springer Berlin Heidelberg, 1991. – ISBN 978–3–540–47471–5, 166–178. – DOI: 10.1007/BFb0035103

[KBR+15] Kemayo, Georges ; Benammar, N. ; Ridouard., Frédéric ; Bauer, Henri ; Richard, Pascal: Improving AFDX end-to-end delays analysis. In: *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, 2015

[Kop95] KOPETZ, H.: TTP/A - A Time-Triggered Protocol for Body Electronics Using Standard UARTS. In: *International Congress and Exposition*, SAE International, feb 1995. – ISSN 0148–7191

[Kop08] KOPETZ, Hermann: The Rationale for Time-Triggered Ethernet. In: *Real-Time Systems Symposium, IEEE International* 0 (2008), S. 3–11. http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/RTSS.2008.33. – DOI http://doi.ieeecomputersociety.org/10.1109/RTSS.2008.33. – ISSN 1052–8725

[Kop11] KOPETZ, H.: *Real-Time Systems - Design Principles for Distributed Embedded Applications*. Springer, 2nd ed., 2011

[Kor92] KORST, Jan H. M.: *Periodic multiprocessor scheduling*, phdthesis, 1992. http://link.springer.com/chapter/10.1007/BFb0035103

[KRBR14] KEMAYO, Georges ; RIDOUARD, Frédéric ; BAUER, Henri ; RICHARD, Pascal: A Forward End-to-end Delays Analysis for Packet Switched Networks. In: *22Nd International Conference on Real-Time Networks and Systems*, ACM, 2014 (RTNS '14). – ISBN 978–1–4503–2727–5

[KRR13] KEMAYO, Georges ; RIDOUARD, Henri Frédéric ; RICHARD, Pascal: Optimistic problems in the trajectory approach in FIFO context. In: *18th Conference on Emerging Technologies & Factory Automation, ETFA 2013*, IEEE, 2013. – ISBN 978–1–4799–0864–6 978–1–4799–0862–2, 1–8

[LBT01] LE BOUDEC, Jean-Yves ; THIRAN, Patrick: *Network calculus: a theory of deterministic queuing systems for the Internet*. Springer, 2001 (Lecture notes in computer science 2050). – ISBN 978–3–540–42184–9

[LCG14] LI, Xiaoting ; CROS, Olivier ; GEORGE, Laurent: The Trajectory approach for AFDX FIFO networks revisited and corrected. In: *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on*, IEEE, 2014, 1–10

[MM04] MINET, Pascale ; MARTIN, Steven: Worst case end-to-end response times for non-preemptive FP/DP* scheduling. Version: 2004. https://hal.inria.fr/inria-00070588/document. 2004. – Forschungsbericht

[MM06a] MARTIN, Steven ; MINET, Pascale: Schedulability analysis of flows scheduled with FIFO: application to the expedited forwarding class. In: *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, IEEE, 2006, 8–pp

[MM06b] MARTIN, Steven ; MINET, Pascale: Worst case end-to-end response times of flows scheduled with FP/FIFO. In: *International Conference on Networking, International Conference on Systems and International Conference on*

Mobile Communications and Learning Technologies (ICNICONSMCL'06), IEEE, 2006, 54–54

[Mok83] Mok, Aloysius K.: *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*, Massachusetts Institute of Technology, Diss., 1983

[MS10] Marouf, Mohamed ; Sorel, Yves: Schedulability conditions for non-preemptive hard real-time tasks with strict period. In: *Proceedings of 18th International Conference on Real-Time and Network Systems, RTNS'10*. Toulouse, France, November 2010

[MS11] Marouf, Mohamed ; Sorel, Yves: Scheduling non-preemptive hard real-time tasks with strict periods. In: *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*, IEEE, 2011, 1–8

[RSF08] Ridouard, Frédéric ; Scharbarg, Jean-Luc ; Fraboul, Christian: Probabilistic upper bounds for heterogeneous flows using a static priority queueing on an AFDX network. In: *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, 2008, 1220–1227

[SR90] Stankovic, John A. ; Ramamritham, Krithi: What is predictability for real-time systems? 2 (1990), Nr. 4, 247–254. http://www.springerlink.com/index/W847111315421W3K.pdf

[SRF02] Scharbarg, Jean-Luc ; Ridouard, Frédéric ; Fraboul, Christian: A Probabilistic Analysis of End-To-End Delays on an AFDX Avionic Network. 5 (2009-02), Nr. 1, 38–49. http://dx.doi.org/10.1109/TII.2009.2016085. – DOI 10.1109/TII.2009.2016085. – ISSN 1551–3203

[Ste10] Steiner, Wilfried: An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks, IEEE, 2010. – ISBN 978–0–7695–4298–0, 375–384

[SV08] Schuster, Teresa ; Verma, Dinesh: Networking concepts comparison for avionics architecture. In: *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, IEEE, 2008, 1–D

[tim16] *Time-Triggered Ethernet*. http://www.sae.org/technical/standards/AS6802. Version: 2016

[TPS15] Tămaş Selicean, Domiţian ; Pop, Paul ; Steiner, Wilfried: Timing Analysis of Rate Constrained Traffic for the TTEthernet Communication Protocol. In: *2015 IEEE 18th International Symposium on Real-Time Distributed Computing (ISORC)*, 2015, S. 119–126

[TTT] TTTech: *AFDX Switch 3U VPX Rugged*. https://www.tttech.com/products/aerospace/flight-rugged-hardware/switches/afdx-switch-3u-vpx-rugged/,

[WW07] Watkins, Christopher B. ; Walter, Randy: Transitioning from federated avionics architectures to integrated modular avionics. In: *Digital Avionics Systems Conference, 2007. DASC'07. IEEE/AIAA 26th*, IEEE, 2007, 2–A

[ZLX+13] Zhao, Luxi ; Li, Qiao ; Xiong, Ying ; Zheng, Zhong ; Xiong, Huagang: Using multi-link grouping technique to achieve tight latency in network calculus. In: *2013 IEEE/AIAA 32nd Digital Avionics Systems Conference (DASC)*, IEEE, 2013, 2E3–1

Summary

Buffer Analysis and Message Scheduling for Real-Time Networks

For real-time systems, not only is the correct value of the computation (logic correctness) important, but also the point in time when this value is available (temporal correctness). In distributed real-time systems, logical and temporal correctness depend on the interaction of the connected computer nodes. Therefore, communication in distributed real-time systems must be done via a real-time communication system. This type of communication system should offer low message transport latency and low jitter, and thus cause as little impact as possible on the response time of the tasks.

As described by Decotignie in [Dec05], Ethernet [iee12] is fast, easy to install, and the interface ICs are cheap. However, Ethernet does not offer the temporal guarantees required by a real-time communication system. In order to close this gap, research groups from academia and industry have proposed a number of solutions. Two of these solutions are: Avionics Full-Duplex Switched Ethernet (AFDX) [AFD09] and Time-Triggered Ethernet [tim16].

In this dissertation we address two problems faced during the design of AFDX and TTEthernet: data loss due to buffer overflow in AFDX networks with multiple priority traffic, and generation of schedules for TT-frames in TTEthernet networks. Our contributions in this dissertation solve these two problems. We solve the first problem by proposing a method to compute an upper bound for the backlog of each buffer in an AFDX network with multiple priority traffic. We solve the second problem by proposing an off-line scheduler for the TT-frames of TTEthernet.

Chapter I – Introduction

We start this chapter with a brief introduction to the areas of research explored in this dissertation. Additionally, we present the problems addressed and the contributions of this dissertation. We conclude this chapter with the overview of the dissertation structure.

Chapter II – AFDX Background and State-of-the-Art in Buffer Backlog Analysis

In this chapter, we present the fundamental properties of AFDX networks and the state-of-the-art approaches to compute upper bounds for end-to-end latency and buffer backlog. We show in Section II.1, that AFDX networks can ensure predictable temporal properties for frames traversing the network by means of AFDX-compliant switches and end-systems. We describe the concept of *virtual link* and show how AFDX can provide bandwidth isolation. Section II.1 further describes the *contention* issue in AFDX networks, i.e., the condition when multiple frames try to egress an output port at the same time. AFDX addresses the contention issue by buffering, which in turn raises two important questions presented in Section II.2: What is the largest latency encountered by a frame due to buffering? And, what is the largest backlog in an output buffer?

In Section II.3 we present the state-of-the-art methods to address these two questions. Due to the unsynchronized nature of AFDX networks and the large number of virtual links in commercial airplane's avionics systems, simulating all possible combinations of frames dispatching times is intractable. Nevertheless, we show in Section II.3.4, that simulation is used in small networks to estimate the pessimism of other approaches. Therefore, a number of methods have been proposed to compute upper bounds for the worst case traversal time and largest buffer backlog. We divide the presentation of these existing methods into two categories: the first, with methods that analyze the traffic at flow level and bound the traffic and services on the network by means of *envelopes*; the second, with methods that analyze the network at a finer granularity by investigating the effects of the network service and the cross-traffic on a given frame. For most of the compared cases (see [BSF12b]), methods of the second category lead to the computation of tighter upper bounds when compared to those achieved by methods of the first category (including network calculus with serialization).

We show in Section II.3.5 that model checking has been used to compute the exact worst case end-to-end latency in AFDX networks. This method however, does not does not scale and is therefore not used for the upper bound computations in large avionics networks. Yet, similar to simulation methods presented in Section II.3.4, this approach can be used to evaluate the pessimism of the upper bounds achieved by other methods.

Despite the importance of avoiding buffer overflows in AFDX networks, most work on upper bound computation targets the worst case end-to-end latency, not giving much attention to the buffer backlog analysis. We show in this chapter that, the state-of-the-art AFDX buffer backlog analysis does not provide a method to compute deterministic upper bounds for buffer backlog of AFDX networks with multiple priority traffic.

Chapter III – Buffer Backlog Upper Bound for AFDX Networks with Multiple Priority Traffic

In this chapter, we present a method to compute an upper bound for the backlog of every buffer on an AFDX network. We show that contention occurs on a switch output port every time multiple frames try to egress this port at the same time. The contention issue is addressed by means of buffering in AFDX networks. Considering safety-critical nature of applications in avionics systems, overflow of these buffers must be avoided at all cost. Even though the AFDX standard recommends a minimum amount of memory

to be reserved for the output buffers, this amount does not represent a safe bound for every network configuration. Further, this recommendation does not specify how the reservation should be shared among buffers of different priorities.

The current AFDX standard allows for the classification of network traffic in two priorities: *high* and *low*. However, some existing commercial solutions extended the number of used priorities. In the approach presented in this chapter, we assumed an AFDX network in which virtual links are classified with any number of priority levels, and provided the computation of an upper bound for each buffer (one per priority level) in the network.

For each output buffer, we investigate the largest backlog encountered by probe frames of each virtual link with the same priority as that buffer. Our method is based on the largest busy period encountered by a frame of each virtual link that egresses a switch through the output port in which the buffer under analysis belongs to. We show that the buffer backlog depends not only on the static properties (amount, size and priority) of the of the frames in the busy period: also the order and the point in time in which they ingress and egress the buffers impact on the buffer backlog computation.

In order to simplify the analysis of the buffer backlog of AFDX networks with multiple priorities, we introduce the concept of *type of interval*. The idea behind type of intervals is to analyze the ingress and egress of frames in the largest busy period, and identify intervals of time with the same properties of one of the *types of interval*. Any time interval that does not match the properties defined by our four *types of interval* can be decomposed into smaller intervals until each of them adheres to properties of one of these types. We show that, according to the defined properties, only four types of intervals exist.

We use the four types of interval to describe the properties of the scenario (ingress and egress order of frames in the largest busy period) leading to the largest backlog encountered by a frame of a virtual link. We call this scenario *worst case scenario*. We show that, in the worst case scenario, the probe frame is the last arriving competing frame. Three effects might delay the dispatch of the probe frame and consequently increase the backlog faced by it: blocking (due to lower priority frames), interference (due to higher priority frames) and idle time on the output link. The worst case scenario is the one in which the largest sum of these three effects occur.

Once the worst case scenario is identified, we compute an upper bound for the buffer backlog encountered by a frame of *one* and then of *all* virtual links that egresses a switch through the output port in which the buffer under analysis belongs to. This computation provides an upper bound for the backlog of the buffer under analysis.

We further discuss how the results achieved by our method can be applied to AFDX switches that do not conform with our assumptions on how the ingress and egress of frames are handled by AFDX switches.

Our experiments showed that the computational cost of our method is negligible when compared to the cost of computing the end-to-end delay using the trajectory approach.

Chapter IV – TTEthernet Background and State-of-the-Art in Time-Triggered Schedulers

Time-triggered Ethernet (TTEthernet) is an extension of the switched Ethernet (IEEE 802.3) which allows for Best-Effort (BE), Rate-Constrained (RC) and Time-Triggered (TT) traffic to coexist in the same communication system In this chapter, we present the fundamental properties of TTEthernet required to understand our scheduler proposed in Chapter V. Additionally, we present in this chapter the state-of-the-art in the generation of a schedule for TT virtual links in TTEthernet.

We start this chapter by presenting our terminology used to address the data transmission over TTEthernet. Then, we show how the TTEthernet clock synchronization protocol impacts on the scheduling of TT-frames. In TTEthernet, scheduling a TT virtual link means reserving a time window (transmission windows) at a scheduled point in time for the transmission of each frame of that virtual link on each physical link in its path. In Section IV.3.2, we show the similarities of the problem of scheduling TT-frames in a physical link and the problem of scheduling non-preemptive strictly periodic jobs in offset1 free systems (or independent operations strictly periodically).

A valid TT schedule for a given physical link shall not contain overlapping transmission windows, and therefore no contention among TT-frames. Nevertheless, contention might occur due to the transmission of BE- and/or RC-frames, since their transmission times are not known in advance. We present in Section IV.3.3 how TTEthernet addresses contention.

We depict in Section IV.3.5 how the end-to-end latency of a frame varies according to the schedule on each physical link along the path of a virtual link, and presented how to reach the minimum latency. And in Section IV.3.6, we describe a method to account for the temporal requirements of RC virtual links during the generation of a TT scheduling table.

Section IV.4 presents the state-of-the-art approaches in the generation of scheduling tables for TTEthernet.

Chapter V – Off-Line Scheduler for Time-Triggered Networks

In this chapter, we propose an off-line scheduler for Time-Triggered Ethernet called Search Tree based Scheduler for Time-Triggered Networks (STSTTN). The goal of STSTTN is to generate a schedule for the transmission of TT-frames in each physical link in the network, such that the time windows reserved for the transmission of TT-frames do not overlap in any physical link. We call *transmission window*, the timespan reserved for the transmission of a TT-frame. A *phase* represents the start time of the transmission window used to transmit the first frame of a virtual link.

We model this scheduling problem as a *search tree*, in which each virtual link is assigned to a level of the search tree. Each *vertex* in a level of the search tree represents a set with phases that can be used to schedule the virtual link on that level. Each *edge* represents a phase. The *path* leading to each vertex describes a schedule. STSTTN traverses the search tree until it reaches a leaf. If it reaches a valid leaf, the schedule for all virtual links traversing a physical link is found. If, on the contrary, it reaches an invalid leaf, the scheduler backtracks. If no further backtracking is possible, the set of

virtual links is deemed unschedulable.

We present two approaches to decrease the search space of the search tree called *look back* and *look ahead*. Both approaches compare, pairwise, the properties of virtual links scheduled along the visited path and compute a set of possibly feasible phases for the next virtual link. The difference between *look back* and *look ahead* is that the latter, additionally checks for the properties of virtual links which have not been scheduled yet. We present an example showing that, even in a small set of virtual links, *look back* and *look ahead* can decrease the search space by one order of magnitude (14 vertices in Figure V.5 instead of 170 vertices in Figure V.3). We demonstrate that other criteria, e.g., the algorithms to assign virtual links to levels, and to select the edges, also impact on the search space.

We show that selecting a "bad" edge at upper levels of the tree might lead to infeasible schedules which can only be detected at deeper levels, leading to a large number of backtracks before the scheduler deems the vertex invalid. In order to allow for STSTTN to earlier discard such "bad" vertices, we present a preventive tree pruning heuristic.

Traversing the search tree and backtracking imply different computational costs depending on the approach used to construct the search tree. For the *look back* approach, traversing and backtracking computational costs increase at deeper levels of the search tree. The opposite is true for the *look ahead* approach: traversing and backtracking computational costs decrease at deeper levels of the search tree.

The schedule of a virtual link, requires the selection of a phase in each traversed physical link. We show that, in order to achieve the minimum transmission latency, a transmission window must start immediately after the frame is ready for transmission. We present an approach to schedule the TT virtual links which allows for the minimum transmission latency.

Chapter VI

This chapter summarizes the main contributions of this dissertation and describes future work.

Appendix A

We present in this appendix, details of the proof of Theorem III.1.

Appendix B

We present in this appendix, how to use the trajectory approach to compute these competing frames.

Zusammenfassung

Puffer Analyse und Nachrichten Scheduling für Echtzeitnetzwerke

Für Echtzeitsysteme ist nicht nur der korrekte Wert der Berechnung (logische Korrektheit) wichtig, sondern auch der Zeitpunkt, zu dem dieser Wert verfügbar ist (zeitliche Korrektheit). In verteilten Echtzeitsystemen hängt die logische und zeitliche Korrektheit von der Interaktion der verbundenen Computerknoten ab. Die Kommunikation in verteilten Echtzeitsystemen erfolgt daher über ein Echtzeit-Kommunikationssystem. Diese Art von Kommunikationssystem sollte eine niedrige Nachrichtentransportlatenz mit möglichst niedriger Varianz bieten und somit die Reaktionszeit der laufenden Prozesse so wenig wie möglich beeinflussen.

Ethernet [iee12] ist, wie von Decotignie in [Dec05] beschrieben, schnell und einfach zu installieren. Außerdem sind die Schnittstellen-Bausteine günstig. Ethernet bietet jedoch nicht die zeitlichen Garantien, die ein Echtzeit-Kommunikationssystem benötigt. Um diese Lücke zu schließen, haben Forschergruppen aus Wissenschaft und Industrie eine Reihe von Lösungen vorgeschlagen. Zwei dieser Lösungen sind: Avionics Full-Duplex Switched Ethernet (AFDX) [AFD09] und Time-Triggered Ethernet [tim16].

Diese Dissertation befasst sich mit zwei Problemen, die beim Design von AFDX und TTEthernet auftreten: Zum einen das Problem von Datenverlust durch Pufferüberlauf in AFDX-Netzwerken mit mehr als zwei Prioritätsstufen für Datenverkehr und das Problem der Erstellung eines Zeitplans für TT-Frames in TTEthernet-Netzwerken. Die Beiträge in dieser Dissertation lösen diese beiden Probleme. Zur Lösung des ersten Problems wird eine Methode vorgeschlagen, um eine Obergrenze für den Rückstand jedes Puffers in einem AFDX-Netzwerk mit mehr als zwei Prioritäten für Datenverkehr zu berechnen. Das zweite Problem wird mit dem Einsatz eines Offline-Scheduler für TT-Frames von TTEthernet gelöst.

Chapter I – Introduction

Dieses Kapitel beginnt mit einer kurzen Einführung in die in dieser Dissertation untersuchten Forschungsgebiete. Darüber hinaus werden die angesprochenen Probleme und Beiträge dieser Dissertation vorgestellt. Dieses Kapitel schließt mit einem Überblick über die Gliederung dieser Dissertation.

Chapter II – AFDX Background and State-of-the-Art in Buffer Backlog Analysis

Dieses Kapitel stellt die grundlegenden Eigenschaften von AFDX-Netzwerken und die derzeitigen Ansätze zur Berechnung von Obergrenzen für Gesamtlatenz und Pufferrückstände vor. In Abschnitt II.1 wird gezeigt, dass AFDX-Netzwerke vorhersagbare zeitliche Eigenschaften für Frames gewährleisten können, die das Netzwerk mit der Hilfe AFDX-kompatibler Switches durchlaufen. Das Konzept von virtuellen Kommunikationskanälen wird beschrieben und es wird aufgezeigt, wie AFDX Bandbreitenisolation garantieren kann. Des Weiteren beschreibt Abschnitt II.2 das Problem "Konfliktsituation" in AFDX-Netzwerken, d.h. die Situation, wenn mehrere Pakete gleichzeitig versuchen, einen Ausgangsport zu nutzen. AFDX adressiert das Konfliktproblem durch Pufferung, was wiederum zwei wichtige Fragen aufwirft, die in Abschnitt [sec: prop] vorgestellt werden: Was ist die größte Latenz, der ein Paket aufgrund von Pufferung ausgesetzt ist und was ist der größte Rückstand in einem Ausgangspuffer?

In Abschnitt II.3 werden die bestehenden Methoden vorgestellt, um die zuvor genannten Probleme zu lösen. Aufgrund der unsynchronisierten Natur von AFDX-Netzwerken und der großen Anzahl von virtuellen Kommunikationskanälen in den Avioniksystemen kommerzieller Flugzeuge ist die Simulation aller möglichen Kombinationen von Sendezeitpunkten für Pakete schwer zu bewältigen. Trotz alledem zeigt Abschnitt II.3.4, dass Simulationen für kleine Netzwerke verwendet werden, um die Genauigkeit anderer Ansätze abzuschätzen.Aus diesem Grund wurde eine Reihe von Methoden vorgeschlagen, um Obergrenzen für die höchste Gesamtlatenz und den größten Pufferrückstand zu berechnen. Die Beschreibung dieser existierenden Methoden wird in zwei Kategorien aufgeteilt: Die erste Kategorie zeigt Methoden, die den Datenverkehr auf Datenstromebene analysieren und mit Obergrenzen für die Anforderungen an das Netzwerk und Untergrenzen für die bereitgestellte Leistung des Netzwerks arbeiten.Die zweite Kategorie behandelt Methoden, die das Netzwerk detailliert analysieren, indem die Auswirkungen der Netzwerkleistung und des übrigen Datenverkehrs auf einzelne Pakete untersucht werden. Für meisten der betrachteten Fälle (siehe [BSF12b]) führen Methoden der zweiten Kategorie zu niedrigeren Obergrenzen im Vergleich zu Methoden der ersten Kategorie (einschließlich Netzwerk-Kalkulus mit Serialisierung).

Abschnitt II.3.5 zeigt, dass Modellprüfung zur Berechnung der höchsten Gesamt-Latenz in AFDX-Netzwerken verwendet wurde. Diese Methode ist jedoch nicht skalierbar und wird daher in großen Avioniknetzwerken nicht zur Berechnung von Obergrenzen verwendet. Ähnlich wie bei den in Abschnitt II.3.4 vorgestellten Simulationsmethoden kann dieser Methode dazu verwendet werden, um die Genauigkeit der Obergrenzen anderer Methoden zu bewerten. Obwohl es wichtig ist, Pufferüberläufe in AFDX-Netzwerken zu vermeiden, konzentriert sich ein Großteil der bestehenden Forschung auf die Berech-

nung der Obergrenze für die Worst-Case Gesamt-Latenz, wobei der Analyse der Pufferrückstände wenig Beachtung geschenkt wird. Dieses Kapitel zeigt, dass die moderne AFDX-Pufferrückstandsanalyse keine Methode zur Berechnung von deterministischen Obergrenzen für Pufferrückstände von AFDX-Netzwerken mit mehr als zwei Prioritäten für Datenverkehr liefert.

Chapter III – Buffer Backlog Upper Bound for AFDX Networks with Multiple Priority Traffic

Dieses Kapitel stellt eine Methode vor, um eine Obergrenze für den Rückstand jedes Puffers in einem AFDX-Netzwerk zu berechnen. Das Auftreten einer Konfliktsituation wenn zwei Pakete versuchen denselben Switch-Ausgangsport zu verlassen wird illustriert. Die Konfliktsituation wird mittels Pufferung in AFDX-Netzwerken adressiert. In Anbetracht der sicherheitskritischen Natur von Anwendungen in Avioniksystemen muss ein Überlauf dieser Puffer unbedingt vermieden werden. Obwohl der AFDX-Standard empfiehlt, dass für die Ausgabepuffer ein Mindestmaß an Speicherplatz reserviert wird, stellt dies keine sichere Grenze für jede Netzwerkkonfiguration dar. Außerdem wird in dieser Empfehlung nicht angegeben, wie dieser Speicherplatz zwischen Puffern mit unterschiedlichen Prioritäten aufgeteilt werden sollte.

Der aktuelle AFDX-Standard ermöglicht die Klassifizierung des Netzwerkverkehrs in zwei Prioritäten: *high* und *low*. Einige bestehende kommerzielle Lösungen erweiterten bereits die Anzahl der verwendeten Prioritäten. In dem in diesem Kapitel vorgestellten Ansatz wird ein AFDX-Netzwerk angenommen, in dem virtuelle Links mit einer beliebigen Anzahl von Prioritätsstufen klassifiziert werden können und die Berechnung einer Obergrenze für jeden Puffer (einen pro Prioritätsstufe) im Netzwerk beschrieben.

Für jeden Ausgangspuffer wird der größtmögliche Rückstand ermittelt, den Pakete jedes virtuellen Kommunikationskanals ,welcher dieselbe Priorität wie dieser Puffer haben, durchlaufen. Diese Methode basiert auf der längsten Busy Period, die von einem Paket eines virtuellen Kommunikationskanals angetroffen wird, der einen Switch durch den Ausgangsport verlässt, zu dem der zu analysierende Puffer gehört. Der Pufferrückstand hängt nicht nur von den statischen Eigenschaften (Menge, Größe und Priorität) der Pakete in der Busy Period ab: Auch die Reihenfolge und der Zeitpunkt, in dem sie in den Puffer ein- und austreten, beeinflussen die Berechnung des Pufferrückstands.

Um die Analyse des Pufferrückstands von AFDX-Netzwerken mit mehreren Prioritäten zu vereinfachen, wird das Konzept des *Intervalltyps* eingeführt. Die Idee hinter den Intervalltypen besteht darin, den Eingang und den Ausgang von Frames in der längsten Busy Period zu analysieren und den Zeitintervallen mit denselben Eigenschaften einen Intervalltyp zuzuordnen. Jedes Zeitintervall, das keinem Intervalltyp zugeordnet werden kann, kann in kleinere Intervalle zerlegt werden, bis jedes von ihnen den Eigenschaften eines dieser Typen entspricht. Es wird gezeigt, dass gemäß den definierten Eigenschaften nur vier Intervalltypen existieren.

Die vier Intervalltypen werden verwendet, um das Szenario (Eingangs- und Ausgangsreihenfolge von Paketen in der längsten Busy Period) zu beschreiben, welches zu dem größten Rückstand führt, den ein Paket eines virtuellen Kommunikationskanals antrifft. Dieses Szenario wird das *worst case scenario* genannt. In diesem Worst-Case Szenario ist das analysierte Paket das letzte der ankommenden, konkurrierenden Pakete. Drei Ef-

fekte können den Sendevorgang eines Pakets verzögern und folglich den Pufferrückstand erhöhen: Blockierung (aufgrund von Paketen mit niedrigerer Priorität), Interferenz (aufgrund von Paketen mit höherer Priorität) und Leerlaufzeit auf der Ausgangsverbindung. Das Worst-Case-Szenario ist dasjenige, bei dem diese drei Effekte maximiert werden.

Sobald das Worst-Case-Szenario identifiziert ist, wird eine Obergrenze für den Pufferrückstand berechnet,zunächst für die Pakete eines einzelnen virtuellen Kommunikationskanals, und anschließend für die Pakete aller virtuellen Kommunikationskanäle die den analysierten Puffer durchlaufen. Diese Berechnung liefert eine Obergrenze für den Rückstand des analysierten Puffers. Weiterhin wird untersucht, wie die Ergebnisse dieser Methode auf AFDX-Switches angewendet werden können, die nicht unseren Annahmen über das Empfangen und Senden von Paketen entsprechen

Chapter IV – TTEthernet Background and State-of-the-Art in Time-Triggered Schedulers

Time-triggered Ethernet (TTE) ist eine Erweiterung des Switched Ethernet (IEEE 802.3), die es Datenströmen der Klassen Best-Effort (BE), Rate- Constrained (RC) und Time- Triggered (TT) erlaubt, im selben Netzwerk zu co-existieren. Dieses Kapitel stellt die grundlegenden Eigenschaften von TTE vor, die zum Verständnis des in Kapitel V vorgeschlagenen Schedulers erforderlich sind. Darüber hinaus wird der Stand der Technik bei der Erstellung eines Ablaufplans für virtuelle TT-Links in TTE vorgestellt.

Dieses Kapitel beginnt mit der Terminologie, die verwendet wird, um die Datenübertragung über TTE zu beschreiben. Es wird gezeigt, wie sich das TTE-Taktsynchronisationsprotokoll auf das Scheduling von TT-Pakets auswirkt. In TTE ist das Scheduling eines virtuellen TT-Kanals gleichbedeutend mit der Reservierung eines Zeitfensters (Übertragungsfensters) zu einem geplanten Zeitpunkt für die Übertragung jedes Pakets eines virtuellen Kommunikationskanals auf jeder physikalischen Verbindung auf dessen Pfad. Abschnitt IV.3.2 zeigt die Ähnlichkeiten zwischen dem Scheduling-Problem von TT-Paketen in einer physischen Verbindung und dem Scheduling-Problem von nichtpreemptiven streng periodischen Jobs in offset-freien Systemen (oder unabhängigen streng periodischen Operationen).

Ein gültiger TT-Zeitplan für eine bestimmte physikalische Verbindung darf keine überlappenden Übertragungsfenster und daher keine Konflikte zwischen TT-Paketen enthalten. Trotzdem kann ein Konflikt aufgrund der Übertragung von BE- und / oder RC-Paketen auftreten, da deren Übertragungszeiten nicht im Voraus bekannt sind. Abschnitt IV.3.3 zeigt, wie TTE diese Konflikte löst.

Abschnitt IV.3.5 stellt dar, wie sich die Gesamt-Latenz eines Pakets je nach Zeitplan auf den physischen Verbindungen entlang des Pfades eines virtuellen Kommunikationskanals ändert und wie man die minimale Latenz erreicht. Abschnitt IV.3.6 beschreibt eine Methode, um die zeitlichen Anforderungen von RC-Kommunikationskanälen während der Generierung einer TT-Schedulingtabelle zu berücksichtigen.

Abschnitt IV.4 zeigt die bestehenden Ansätze bei der Erstellung von Schedulingtabellen für TTE.

Chapter V – Off-Line Scheduler for Time-Triggered Networks

In diesem Kapitel wird ein Offline-Scheduler für Time-Triggered Ethernet (TTE) beschrieben - genannt Search Tree-based Scheduler for Time-Triggered Networks (STSTTN). Das Ziel von STSTTN besteht darin, einen Zeitplan für die Übertragung von TT-Frames in jeder physikalischen Verbindung im Netzwerk zu erzeugen, so dass sich die für die Übertragung von TT-Paketen reservierten Zeitfenster in keiner physikalischen Verbindung überlappen. Diese für die Übertragung eines TT-Pakets reservierte Zeitspanne wird Übertragungsfenster genannt. Eine *Phase* stellt die Startzeit des Übertragungsfensters dar, das zum Senden des ersten Pakets eines virtuellen Kommunikationskanals verwendet wird.

Dieses Scheduling-Problem wird als Suchbaum modelliert, in dem jeder virtuelle Kommunikationskanal einer Ebene des Suchbaumes zugeordnet ist. Jede Kante stellt eine Phase dar. Der Pfad durch den Suchbaum, der zu jedem Knoten führt, beschreibt einen möglichen Zeitplan. STSTTN durchläuft den Suchbaum, bis er ein Blatt erreicht. Der Zeitplan ist gefunden, wenn ein gültiges Blatt erreicht wurde und somit alle virtuellen Kommunikationskanäle eine physische Verbindung durchlaufen haben. Wenn dagegen ein ungültiges Blatt erreicht wird, beginnt der Scheduler erneut. Wenn kein weiterer Durchlauf möglich ist, wird die Erstellung eines Zeitplans mit dieser Kombination aus virtuellen Kommunikationskanälen als unmöglich angesehen.

Wir stellen zwei Ansätze vor, um den Suchbereich des Suchbaums zu reduzieren. Diese Ansätze heißen *Look Back* und *Look Ahead*. Beide Ansätze vergleichen paarweise die Eigenschaften von virtuellen Kommunikationskanälen, die entlang des bereits besuchten Pfades geplant sind, und berechnen einen Satz von möglicherweise gültigen Phasen für den nächsten virtuellen Kommunikationskanal. Der Unterschied zwischen *Look Back* und *Look Ahead* liegt darin, dass letzterer zusätzlich die Eigenschaften von virtuellen Kommunikationskanälen prüft, die noch nicht geplant wurden. An einem Beispiel wird gezeigt, dass, selbst in einem kleinen Set aus virtuellen Kommunikationskanälen, *Look Back* und *Look Ahead* den Suchraum um eine Größenordnung verkleinern (14 Knoten in Abbildung V.5 statt 170 Knoten in Abbildung V.3). Es wird demonstriert, dass andere Kriterien, z. B. Algorithmen zum Zuweisen virtueller Kanäle zu Suchbaum Ebenen und zum Auswählen der nächsten Kante, ebenfalls den Suchbereich beeinflussen.

Die Auswahl einer ungünstigen Kante in den oberen Ebenen des Baums hat möglicherweise unmögliche Zeitpläne zur Folge, was nur in unteren Ebenen erkannt werden kann. Dies führt zu einer großen Anzahl von Durchläufen, bevor der Scheduler den Knoten für ungültig hält. Damit STSTTN solche Kanten früher verwirft, wird eine vorbeugende Baum-Reduzierungs-Heuristik präsentiert.

Das Durchlaufen des Suchbaumes und die jeweiligen Rückzüge resultieren in unterschiedlichen Berechnungskosten, abhängig vom Ansatz, der verwendet wird, um den Suchbaum zu konstruieren. Für den *Look Back*-Ansatz steigen die Kosten für Durchlauf und Rückzüge bereits auf den unteren Ebenen des Suchbaumes an. Das Gegenteil trifft auf den *Look ahead*-Ansatz zu: Die Berechnungskosten für Durchlauf und Rückzüge nehmen in den unteren Ebenen des Search Trees ab.

Der Zeitplan eines virtuellen Kommunikationskanals erfordert die Auswahl einer Phase in jeder durchlaufenen physikalischen Verbindung. Um eine minimale Übertragungsla-

tenz zu erreichen, muss das Übertragungsfenster sofort starten, sobald das Paket zur Übertragung bereit ist. Es wird ein Ansatz vorgestellt, um die virtuellen TT-Kanäle zu planen, die eine minimale Übertragungslatenz ermöglichen.

Chapter VI

Dieses Kapitel fasst die wichtigsten Beiträge dieser Dissertation zusammen und beschreibt die zukünftige Arbeit.

Appendix A

Der Anhang stellt detailliert den Beweis des Theorems III.1 vor.

Appendix B

In diesem Anhang wird gezeigt, wie die konkurrierenden Frames mit dem Trajectory-Ansatz berechnet werden können.

Rodrigo F. Coelho

Personal details

Name: Rodrigo Ferreira Coelho
Place of birth:Rio de Janeiro, Brazil

Contact

@ dr.rodrigo.f.coelho@gmail.com
✉ Chair of Real Time Systems
TU Kaiserslautern
Postfach 3049
67663, Kaiserslautern

Languages

German (fluent)
English (fluent)
Portuguese (mother tongue)

Education

04.2012 – 11.2017 **PhD in Electrical Engineering**
Technische Universität Kaiserslautern at the Chair of Real-Time Systems, Germany
Title: Buffer Analysis and Message Scheduling for Real-Time Networks

04.2005 – 05.2008 **MSc in Electrical Engineering** major in Automation and Control
Technische Universität Kaiserslautern, Germany
Thesis: Control Theory Approaches to Resource Adaptation in Real-Time
Embedded Systems

07.1996 – 11.2002 **BSc in Electrical Engineering** major in Electronics - 10 semester program
Centro Federal de Educação Tecnológica do Rio de Janeiro CEFET-RJ, Brazil
Thesis: Design and Implementation of a Development Kit for the Microcontroller PIC16F877.

Work Experience

04.2012 – 11.2017 **Wissenschaftlicher Mitarbeiter (Research Assistant), PhD Candidate**
Technische Universität Kaiserslautern, Chair of Real-Time Systems, Germany

10.2008 – 03.2012 **Wissenschaftlicher Mitarbeiter (Research Assistant)**
Technische Universität Kaiserslautern, Chair of Real-Time Systems, Germany

11.2006 – 05.2008 **Wissenschaftliche Hilfskraft (Research student)**
Technische Universität Kaiserslautern, Chair of Real-Time Systems, Germany

11.2005 – 10.2006 **Wissenschaftliche Hilfskraft (Research student)**
Technische Universität Kaiserslautern, Chair of Electronic Design Automation Group, Germany

01.2002 – 12.2004 **Head of the Closed-Circuit Television (CCTV) Department**
3R Telecomunicações LTDA, (Panasonic Telecomunications Distributor in Rio de Janeiro, Brazil)

01.1995 – 07.2000 **Communications Electronics Technician**
3R Telecomunicações LTDA, (Panasonic Telecomunications Distributor in Rio de Janeiro, Brazil)

Exchange Programs

02.2001 - 08.2001 **Bachelor Internship**
BMW AG, Department for Electronic Control Unit Development (M-Engines Electronic), Munich, Germany

08.2000 - 10.2000 **DAAD-CAPES Funded Exchange Program**
Fachhochschule Munich, Germany