RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Doctorat de l'Université de Toulouse

délivré par l'ISAE-SUPAERO

A Methodology for Credit-Based Shaper Deployment in Time Sensitive Networks: Design, Analysis and Optimization

---

Thèse présentée et soutenue, le 19 février 2026 par

Santiago TORRES BORDA

École doctorale
EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse

Spécialité
Informatique et Télécommunications

Unité de recherche
ISAE-ONERA MOIS MOdélisation et Ingénierie des Systèmes

Thèse dirigée par
Ahlem MIFDAOUI

Composition du jury
Mme Christine ROCHANGE, Présidente, Examinatrice, Université de Toulouse
M. Saad MUBEEN, Rapporteur, Mälardalen University
M. Frédéric RIDOUARD, Rapporteur, ISAE-ENSMA
M. Frank SINGHOFF, Examinateur, Université de Brest
Mme Anaïs FINZI, Examinatrice, TTTech Computertechnik AG
Mme Ahlem MIFDAOUI, Directrice de thèse, ISAE-SUPAERO

A Methodology for Credit-Based Shaper Deployment in Time Sensitive Networks: Design, Analysis and Optimization

Santiago Torres Borda

A thesis submitted for the degree of
Doctor of Philosophy

Département d'Ingénierie pour les Systèmes Complexes (DISC)
ISAE-SUPAERO

May 18, 2026

Abstract — Real-time systems in domains such as aerospace, automotive and industrial automation are increasingly migrating toward Ethernet-based Time-Sensitive Networking (TSN) to meet stringent timing and reliability requirements. While mechanisms like the Credit-Based Shaper (CBS), offer a balance between determinism and complexity, existing literature predominantly assumes a *Full CBS Deployment*, where every network switch is TSN-capable. This assumption overlooks the economic and practical reality of heterogeneous networks that must integrate legacy Ethernet devices. Furthermore, the interaction between CBS and reliability mechanisms, specifically Frame-Replication and Elimination for Reliability (FRER), remains under-researched, despite evidence that FRER significantly increases traffic burstiness, which is specifically addressed by CBS.

This thesis proposes a novel methodology for the *Partial CBS Deployment* in time-sensitive networks. We formulate the deployment strategy as an optimization problem aimed at minimizing the number of TSN-enabled devices while ensuring worst-case schedulability requirements using Network Calculus (NC). We introduce a heuristic algorithm that strategically places and configures CBS instances to mitigate traffic burstiness only where necessary, ensuring interoperability with legacy infrastructure. The proposed methodology is validated through a representative industrial use-case, where schedulability constraints were respected while reducing implementation costs.

The methodology is further extended to be FRER-aware, addressing the burstiness increase introduced by such a mechanism. This extended approach neutralizes the adverse effects of redundancy on network latency while minimizing the use of TSN-enabled devices. This extension is validated through the same representative industrial use-case used for the base version of our methodology (FRER-blind). Here, the observations were consistent with our findings for the original approach.

Finally, the thesis evaluates the optimality as well as the scalability of our proposed methodology through synthetic use-cases. We first benchmarked our approach against an exhaustive *Brute Force* exploration to assess its optimality. Insights from this comparison led to the development of an improved CBS deployment heuristic designed for better optimality. Lastly, the scalability of such an improved approach was performed. The results confirmed that this enhanced solution consistently approximates the optimal, minimal-cost configuration while remaining computationally efficient under network scale increase.

Keywords: TSN, CBS, FRER, NC, Interoperability

Résumé — Les systèmes temps réel dans des domaines tels que l'aérospatiale, l'automobile et l'automatisation industrielle migrent de plus en plus vers les réseaux Ethernet Time-Sensitive Networking (TSN) pour répondre à des exigences strictes de temporalité et de fiabilité. Bien que des mécanismes comme le Credit-Based Shaper (CBS) offrent un équilibre entre déterminisme et complexité, la littérature existante suppose majoritairement un «déploiement complet du CBS », où chaque commutateur du réseau est compatible TSN. Cette hypothèse néglige la réalité économique et pratique des réseaux hétérogènes qui doivent intégrer des équipements Ethernet traditionnels. De plus, l'interaction entre le CBS et les mécanismes de fiabilité, spécifiquement la réplication et l'élimination de trames pour la fiabilité (Frame Replication and Elimination for Reliability (FRER)), reste peu étudiée, malgré le fait avéré que le FRER augmente considérablement les rafales de trafic, problème spécifiquement traité par le CBS.

Cette thèse propose une nouvelle méthodologie pour le «déploiement partiel du CBS », dans les réseaux sensibles au temps. Nous formulons la stratégie de déploiement comme un problème d'optimisation visant à minimiser le nombre d'équipements compatibles TSN tout en garantissant les exigences d'ordonnançabilité au pire cas à l'aide du Calcul Réseau (NC). Nous introduisons une approche heuristique qui place et configure stratégiquement les instances de CBS pour atténuer les rafales de trafic uniquement là où cela est nécessaire, assurant ainsi l'interopérabilité avec l'infrastructure existante. Le cadre proposé est validé au travers d'un cas d'usage industriel représentatif, où nous avons observé le respect des contraintes d'ordonnançabilité couplé à une réduction des coûts.

Cette méthodologie est ensuite étendue pour prendre en compte le FRER (FRER-aware), traitant l'augmentation des rafales introduite par ce mécanisme. Cette approche étendue neutralise les effets de la redondance sur la latence du réseau, afin de minimiser l'utilisation d'équipements compatibles TSN. Cette extension est validée par le même cas d'usage industriel représentatif utilisé pour la version originelle de la méthodologie. Ici, les observations sont cohérentes avec celles relevées pour l'approche originelle.

Enfin, la thèse évalue l'optimalité et le passage à léchelle de l'approche proposée à travers des cas d'usage synthétiques. Nous avons d'abord comparé notre approche à une exploration exhaustive par «force brute»pour évaluer son efficacité économique. Les conclusions tirées de cette comparaison ont conduit au développement d'une méthodologie de déploiement améliorée conçue pour une meilleure optimalité. Enfin, l'évaluation du passage à l'échelle de l'approche améliorée a été réalisée. Ces tests ont confirmé que cette solution améliorée s'approche systématiquement de la configuration optimale à coût minimal tout en restant efficace en termes de complexité quand la taille du réseau augmente.

Mots clés: TSN, CBS, FRER, Calcul Réseau, Interopérabilité

Acknowledgements

I would like to open this document by recognizing the members of my dissertation jury. I owe a particular debt to my reviewers, Frédéric Ridouard and Saad Mubeen, for the meticulous attention they gave to this work; their insightful critiques were instrumental in refining the quality of this document. I am also honored to have Anaïs Finzi, Christine Rochange, and Frank Singhoff serve as members of my jury. I appreciate their time, their willingness to engage with my research, and the honor of their presence in this final stage of my Ph.D.

At the heart of this academic journey was the guidance of my supervisor, Ahlem Mifdaoui, who stood by me through the challenging journey of completing this thesis. I am moved by your mentorship and by the way you ensured I always had the best possible opportunities to learn and grow.

I have also been incredibly fortunate to work within the supportive environment of the SYSCO team. My heartfelt appreciation goes to Oana, without whom I would never have known about the possibility of pursuing my Ph.D. at ISAE-SUPAERO; to Marina, for the insightful discussions regarding my future and for supporting me at every turn; and to Jérôme, whose invaluable advice made me feel truly at home at ISAE.

Beyond my home institution, a significant chapter of this work was shaped by my research exchange at the Computer Communications and Applications Laboratory 2 (LCA2) at EPFL. It was a privilege to be mentored by Jean-Yves Le Boudec, who welcomed me into his lab with a kindness that matched his prominence in the field. I am also deeply mindful of the help I received from Ludovic Thomas, who very patiently helped me master the basics of network calculus and helped me initiate my research. My time was further enriched by the chance to learn from brilliant colleagues including Ehsan Mohammadpour, Seyed Mohammadhossein Tabatabaee, and Stéphan Plassart.

Back at ISAE-SUPAERO, I want to recognize the colleagues and friends with whom I shared this path—Alan, Alice, Deborah, Houssam, Julien, Maël, Matthias, Mohammed Ali, Morgan, Thomas, and Valentin—with whom I have had the pleasure of sharing this path. Though our research topics differed, I cherished the activities and rituals we created together; they made the Ph.D. journey a human experience rather than just a scientific one.

On a more personal note, I want to express my complete gratitude to my parents and siblings—Nicolás, Tatiana, Tomás, and Gabriela—who have always supported and believed in me. Your kind words and unwavering encouragement helped me remain resilient throughout this journey.

To Marta: My deepest gratitude belongs to you. Through every triumph and every challenge, your unwavering presence has been my anchor. Thank you for your endless patience, for always choosing to listen, and for supporting me unconditionally. More than anything, thank you for believing in me, and for lending me that belief when I struggled to find my

own.

Finally, I reach out to my friends, both near and far, whose company I have enjoyed deeply over the last three years. Whether it was a quick coffee or a long-distance call, your friendship was a vital source of strength.

Contents

List of Acronyms **xv**

List of Publications **xvii**

1 Introduction **3**

1.1 Context and Motivation 3

1.2 Problem Statement 4

1.3 Research Objectives 4

1.4 Dissertation Contributions and Outline 4

2 Background and State of the Art **7**

2.1 Introduction 7

2.2 Real-time networking Requirements and Evolution 8

2.3 TSN versus real-time networking requirements 11

2.4 CBS and State of The Art 17

2.5 Conclusion 25

3 A Cost-Efficient Partial CBS Deployment Framework **27**

3.1 System Model & CBS background 29

3.2 Motivation for Partial CBS Deployment 31

3.3 The Partial CBS Deployment as an Optimization Problem 36

3.4 Framework for Partial CBS Deployment 44

3.5 Framework Validation Over an Automotive Use-Case 55

3.6 Conclusions 60

4 FRER-Aware Partial CBS Deployment **63**

4.1 Extended System Model and Frame Replication and Elimination for Reliability (FRER) Background 64
4.2 Motivation for FRER-aware Partial CBS Deployment 68
4.3 FRER-aware Partial CBS Deployment as an Optimization Problem 71
4.4 Framework for FRER-aware Partial CBS Deployment 75
4.5 Validation over an Automotive Use-Case 83
4.6 Conclusions 87

5 Improved Partial CBS Deployment Framework and Validation 89

5.1 Tools for Optimality and Scalability Evaluation 90
5.2 Burstiness-Aware Partial CBS Deployment Assessment 93
5.3 Assessment of the FRER-Aware Partial CBS Deployment Framework 105
5.4 Conclusions 115

6 Conclusions and Future Work 117

6.1 Conclusions 117
6.2 Future Work 119

A Network Calculus 121

A.1 Mathematical Background: The (min, +) Algebra 122
A.2 Data Modeling and Core Concepts 122
A.3 Arrival Curves 123
A.4 Service Curves 124
A.5 Network Calculus (NC) Basics: The Single Node 125
A.6 Network Analysis: Multiple Nodes 126

B Implementation of the Partial CBS Deployment Framework 131

B.1 Framework Architecture 131
B.2 Solver Class Structure 132

B.3 Verification Client 134

B.4 Usage Guide 135

Bibliography **147**

List of Figures

2.1 Summary of Time-Sensitive Networking (TSN) mechanisms [9] 11

2.2 Bridge operation and TSN mechanisms (adapted from [29]) 12

2.3 Credit evolution of a CBS queue over time 19

3.1 Detailed switch architecture, showing the ingress and egress pipeline of an Output Port. 30

3.2 Illustration of the CBS blocking effect 33

3.3 Topology illustrating the blocking effect of CBS on high-priority shaped traffic. 34

3.4 Impact of IdleSlope on flow f_0 delay 36

3.5 Partial CBS Deployment Framework 44

3.6 Flowchart detailing device selection for burstiness-aware partial CBS placement. 47

3.7 Adapted Industrial Automotive Topology from [102] 56

3.8 Shaped Flows ($p = 0$): Comparison of Full CBS vs. Partial CBS. 59

3.9 Unshaped Flows ($p = 1$): Comparison of Partial CBS vs. Non-Preemptive Static-Priority (NP-SP) (No CBS). 60

4.1 Architectural Overview of FRER operations 65

4.2 Detailed Switch Architecture with FRER Elimination Function 66

4.3 Output arrival curve after the elimination function 69

4.4 Example of a TSN Network with FRER 69

4.5 Overview of the FRER-aware Partial CBS Deployment Framework 76

4.6 FRER-aware device selection strategy for CBS placement 77

4.7 Shaped Flows ($p = 0$): Comparison of deployment strategies. 86

4.8 Unshaped Flows ($p = 1$): Comparison of deployment strategies. 87

5.1 Brute force Comparison Methodology 91

5.2 The topologies used in the experiments, shown in the three different used sizes. 92

5.3 Optimality: Heuristic vs. Brute Force (Var. Size) 94

5.4 Optimality: Heuristic vs. Brute Force (Var. Load) 95

5.5 Optimality Gap across different network sizes under a fixed load condition. . 96

5.6 Optimality Gap across different network loads under a fixed size condition. . 96

5.7 Optimality: Improved Heuristic vs. others (Var. Size) 98

5.8 Optimality: Improved Heuristic vs. others (Var. Load) 99

5.9 Improved Optimality Gap (Variable Load) 100

5.10 Improved Optimality Gap (Variable Size) 100

5.11 Delay bounds comparison (Variable Size) for Shaped Flows $p = 0$ (left column) and Unshaped Flows $p = 1$ (right column). 102

5.12 Delay bounds comparison (Variable Load) for Shaped Flows $p = 0$ (left column) and Unshaped Flows $p = 1$ (right column). 103

5.13 Execution time comparison (Variable Load) 104

5.14 Execution time comparison (Variable Size) 105

5.15 TSN Devices comparison in FRER networks 107

5.16 Improved FRER-aware TSN Devices comparison 109

5.17 Improved FRER-aware optimality gap 110

5.18 Improved FRER-Aware Optimality Gap (Var. Size) 111

5.19 Improved FRER-Aware Optimality Gap (Var. Load) 111

5.20 Improved FRER-aware Shaped delay (Var. Size, $p = 0$) 112

5.21 Improved FRER-aware Shaped delay (Var. Load, $p = 0$) 112

5.22 Improved FRER-aware Unshaped delay (Var. Size, $p = 1$) 113

5.23 Improved FRER-aware Unshaped delay (Var. Load, $p = 1$) 114

5.24 Improved FRER-aware execution time (Var. Load) 114

5.25 Improved FRER-aware execution time (Var. Size) 115

B.1 Component Diagram of the System Architecture. The `CBSDeploymentFramework` interacts with the *Timaeus* verification tool via a TCP socket interface provided by the `TimaeusClient`. 132

B.2 UML Class Diagram of the Network Object Model. The `Network` class aggregates `Device`, `OutputPort`, and `Flow` objects, with specific inheritance for Switches and EndSystems. 133

B.3 UML Class Diagram illustrating the Strategy Pattern used for solver selection. The `CBSDeploymentSolver` context delegates the optimization logic to concrete implementations of `CBSConfig`. 133

B.4 Sequence Diagram showing the optimization workflow. The user script initiates the solver, which iteratively configures the network parameters and validates them via the `TimaeusClient`. 135

List of Tables

3.1 Summary of notations 31

3.2 Flow characteristics for the example network in Figure 3.3. 34

3.3 End-to-end delay bounds for flow f_1 under different CBS deployment scenarios. 35

3.4 Flow Characteristics 51

3.5 Variables of interest to the CBS deployment algorithm from the verification tool 52

3.6 Adapted Flow Characteristics for the Automotive use-case in [102] 57

3.7 Deployed CBS Parameters for the Industrial Use-Case Network 58

4.1 Summary of notations 67

4.2 Flow characteristics for the motivational example network (Figure 4.4). . . . 69

4.3 Impact of FRER on delay bounds 70

4.4 Impact of CBS deployment approaches on delay bounds 71

4.5 Delay bounds for the example network under FRER-aware Partial CBS Deployment algorithm. 81

4.6 Network characteristics for the industrial use case with FRER 84

4.7 Deployed CBS Parameters under FRER-Blind and FRER-Aware heuristics . 85

5.1 Network Size Configurations 92

5.2 Traffic Profile Characteristics 92

5.3 Deployed CBS for the Industrial Use-Case presented in Section 4.4 comparing the original burstiness-aware approach to its improved version 106

5.4 Deployed CBS Parameters under FRER-aware and its improved version for the Industrial Use-Case of Section 4.5 115

List of Acronyms

ADAS Advanced Driver-Assistance Systems

AFDX Avionics Full-DupleX

ATS Asynchronous Traffic Shaping

AVB Audio Video Bridging

BPA Busy Period Analysis

CAN Controller Area Network

CBS Credit-Based Shaper

CPA Compositional Performance Analysis

CQF Cyclic Queuing and Forwarding

FDA Forward End-to-end Delay Analysis

FIFO First-in First-out

FRER Frame Replication and Elimination for Reliability

GCL Gate Control List

HPC High-Performance Computing

ILP Integer Linear Programming

MRA Match Recovery Algorithm

NC Network Calculus

NP-SP Non-Preemptive Static-Priority

PCP Priority Code Point

PEF Packet Elimination Function

PRF Packet Replication Function

PSFP Per-Stream Filtering and Policing

QoS Quality of Service

SRF Sequence Recovery Function

TAS Time-Aware Shaper

TFA Total Flow Analysis

TSA Transmission Selection Algorithm

TSN Time-Sensitive Networking

TT Time-Triggered

UBS Urgency-Based Scheduler

VRA Vector Recovery Algorithm

ZCP Zone Control Processor

List of Publications

The following publications have resulted from the research presented in this thesis:

1. **S. Torres-Borda**, A. Mifdaoui. "Towards a Partial Credit-Based Shaper Deployment Algorithm in Time-Sensitive Networks." In *Proceedings of the 2024 IEEE 20th International Conference on Factory Communication Systems (WFCS)*, Toulouse, France, 2024.

2. **S. Torres-Borda**, A. Mifdaoui. "A Cost-Efficient Credit-Based Shaper Deployment Framework for Time-Sensitive Networks." In *Proceedings of the 2025 28th International Symposium on Real-Time Distributed Computing (ISORC)*, Toulouse, France, 2025.

3. **S. Torres-Borda**, A. Mifdaoui. "FRER-aware Partial CBS Deployment in Time-Sensitive Networks." In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, LNCS, Pisa, Italy, 2025.

4. **S. Torres-Borda**, A. Mifdaoui. "A Feasibility-Driven and Cost-Efficient Framework for Partial CBS Deployment in TSN Networks incorporating FRER". (This work is currently under review)

"If I have seen further it is by standing on the shoulders of Giants."

— Isaac Newton
(attributed originally to Bernard de Chartres)

Chapter 1

Introduction

Real-time embedded systems are currently undergoing a fundamental paradigm shift in their communication architectures. Industries such as automotive, aerospace, and industrial automation are transitioning from proprietary, bandwidth-limited fieldbuses (e.g., Controller Area Network (CAN), ARINC 429) toward high-speed, standardized Ethernet backbones. This transition is driven by the exponential growth in data rates required by modern applications, such as autonomous driving and advanced driver assistance systems (Advanced Driver-Assistance Systems (ADAS)). However, standard Ethernet lacks the deterministic guarantees required for safety-critical communications. To bridge this gap, the TSN task group has developed a suite of standards aimed at providing deterministic latency, reliability, and guaranteed bandwidth over Ethernet networks.

1.1 Context and Motivation

Within the TSN toolset, traffic shaping mechanisms play a pivotal role in enforcing determinism. While Time-Triggered (TT) solutions like the Time-Aware Shaper (TAS) offer strong isolation guarantees, they impose significant architectural complexity, requiring network-wide high-precision synchronization and complex schedule synthesis. In contrast, the CBS, defined in IEEE 802.1Qav, offers a compelling trade-off. It provides bounded latencies and bandwidth guarantees through an asynchronous, credit-based mechanism that does not require global synchronization. This makes CBS an attractive candidate for improving the scalability, modularity, and cost-efficiency of real-time networks.

However, despite the benefits of CBS, its practical deployment faces significant hurdles. Existing research predominantly assumes a *Full CBS Deployment*, where every switch in the network is assumed to be TSN-capable and configured with CBS. This assumption overlooks a critical constraint in industrial reality: the need for cost efficiency and interoperability with legacy infrastructure. Replacing every legacy Ethernet switch with TSN-capable hardware is often prohibitively expensive or logistically infeasible. Consequently, there is a distinct lack of methodologies for *Partial CBS Deployment*, where TSN features are enabled only at strategic points in a network to manage traffic traversing both modern and legacy nodes.

Furthermore, critical networks require high reliability, often achieved through redundancy mechanisms like FRER (IEEE 802.1CB). Current deployment strategies for CBS often ignore the interaction with such protocols. This is a critical oversight, as the use of the FRER

mechanism can introduce an increase in traffic burstiness, potentially impacting the traffic shapers' configurations and benefits.

1.2 Problem Statement

The central problem addressed in this document is the lack of a systematic methodology for the cost-efficient deployment of CBS in Time-Sensitive Networks. Specifically, we identify two primary gaps:

1. **The Legacy Interoperability Gap:** There is no established framework for deploying CBS in networks containing legacy Ethernet switches. Current *all-or-nothing* deployment models fail to optimize resource usage or address the burstiness accumulated by traffic passing through non-shaped, legacy nodes.
2. **The Reliability-Shaping Gap:** The interaction between CBS and redundancy protocols (FRER) is under-researched. Specifically, the impact of frame elimination on traffic burstiness and the subsequent configuration of shapers to mitigate this effect remains an open challenge.

1.3 Research Objectives

To address these challenges, this thesis aims to develop a comprehensive framework for Partial CBS Deployment. We leverage NC as a verification tool to guarantee worst-case performance. The research is guided by three fundamental questions:

- **R.Q.1 (Where):** Where can CBS be effectively deployed to leverage its benefits, while maintaining interoperability with legacy Ethernet devices?
- **R.Q.2 (When):** When is CBS a good solution to improve network performance, considering the different traffic patterns and network topologies?
- **R.Q.3 (How):** How do we achieve CBS integration into a TSN network with other mechanisms potentially affecting it and how CBS parameters, particularly the IdleSlope, can be optimally tuned to balance deterministic guarantees with efficient resource utilization?

1.4 Dissertation Contributions and Outline

This dissertation proposes a novel methodology that treats CBS deployment as an optimization problem: minimizing the number of TSN devices while satisfying hard real-time constraints. The document is organized as follows:

Chapter 2: Background and State of the Art. We review the requirements of real-time networks and the limitations of legacy Ethernet. We analyze the TSN standards, justifying the focus on CBS as a complexity-efficient solution, and highlight the gaps in current literature regarding partial deployment and FRER interaction.

Chapter 3: A Cost-Efficient Partial CBS Deployment Framework. We introduce the *Burstiness-Aware* deployment heuristic. This chapter formulates the partial deployment problem and proposes a strategy to place CBS instances at the first point of contention, effectively mitigating burstiness from legacy nodes. We validate this approach on an industrial use case, demonstrating substantial cost reductions compared to full deployment.

Chapter 4: FRER-Aware Partial CBS Deployment. We extend the system model to include network redundancy. We analyze the burstiness introduced by the FRER elimination function and propose a *FRER-Aware* heuristic. This strategy targets deployment specifically at elimination points, neutralizing protocol-induced burstiness to maintain schedulability. This proposal has been validated on the industrial use case and results show a notable improvement in terms of costs and performance in comparison to the FRER-blind Partial CBS deployment developed in Chapter 3.

Chapter 5: Improved Framework and Validation. We rigorously assess the optimality and scalability of our proposed heuristics against a brute-force baseline using synthetic benchmarks. Based on this analysis, we introduce an *Improved Burstiness-Aware* framework that targets the point of *highest* contention rather than the *first*, significantly reducing the optimality gap.

The chapter then discusses how both the Improved Burstiness-Aware and FRER-Aware heuristics encounter difficulties converging to a solution in FRER-aware scenarios. To address this, we propose a new approach called *Improved FRER-Aware*, which uses a refined method to compute tighter IdleSlopes. Although not necessarily optimal, this demonstrates a significant improvement in success rates over previous methods.

Chapter 6: Conclusion and Future Work. We summarize the findings, answering the research questions and outlining future directions for the optimization of heterogeneous real-time networks.

Chapter 2

Background and State of the Art

Contents

2.1 Introduction 7
2.2 Real-time networking Requirements and Evolution 8
2.3 TSN versus real-time networking requirements 11
2.3.1 Reliability 11
2.3.2 Determinism 12
2.3.3 Modularity 13
2.3.4 Complexity 14
2.3.5 Fairness 15
2.3.6 Costs 15
2.3.7 Discussion 16
2.4 CBS and State of The Art 17
2.4.1 CBS Algorithm 18
2.4.2 State of the Art on CBS 19
2.5 Conclusion 25

2.1 Introduction

Real-time systems are increasingly relying on Ethernet-based communication networks to meet their stringent timing and reliability requirements. For example, within the automotive industry, the transition toward Ethernet-based in-vehicle networks is driven by the necessity for higher bandwidth and flexible communication architectures to support modern applications. Newer applications are being introduced to these systems, especially with the advent of autonomous driving and ADAS. These applications provide functionalities that require real-time constraints to be met, such as Lane Departure Warning, Traffic Sign Recognition, Night Vision, and bird's-eye view systems [1]. Ethernet networks are increasingly being adopted in these systems as traditionally used fieldbuses (e.g., CAN, FlexRay) are unable to meet the high data rate and low latency requirements of these applications [2]. Throughout time, various companies have proposed different solutions to provide real-time capabilities over Ethernet networks, such as EtherCAT [3], ProfiNet [4], TTEthernet [5], and Powerlink [6]. However,

these solutions often require specialized hardware and lack interoperability with other networking protocols due to competition between vendors [2]. This also increases maintenance costs and limits scalability. To address these challenges, the AVNU alliance [7] was formed to promote a standardized version of Ethernet capable of providing real-time guarantees. This initiative led to the integration and further development of the Ethernet Audio Video Bridging (AVB) [8] set of standards, which were later included in the TSN set of standards by the IEEE 802.1 TSN Task Group [9]. A similar trend can be observed in other domains; avionic-oriented networks evolved from fieldbuses such as ARINC 429 and MIL-STD-1553 to Ethernet-based networks, first with Avionics Full-DupleX (AFDX) [10] standardized in ARINC 664 [11], to then consider AVB/TSN integration [12]. Industrial control systems have also followed this path, where fieldbuses are extended to use real-time Ethernet solutions [13], [14].

Thus, the first part of this chapter provides an overview of TSN and discusses how it addresses real-time networking requirements. Subsequently, the mechanisms most appropriate for this context are discussed. This leads to the introduction of the problem statement of this dissertation as well as the research questions we aim to answer, and more particularly concerning CBS. The second part provides a comprehensive analysis of the literature regarding CBS, particularly the performance analysis of real-time networks implementing CBS, as well as the CBS deployment, configuration and interaction with other TSN mechanisms.

2.2 Real-time networking Requirements and Evolution

Based on the literature, real-time network requirements can be classified into six distinct categories [15], [16], [17], [18]: reliability, determinism, modularity, fairness and complexity.

- **Reliability:** Refers to the ability of the network to deliver messages without loss or corruption. This is particularly important in safety-critical applications, where the failure to deliver a message can have severe consequences [19].
- **Determinism:**[1] Refers to the ability to provide formal guarantees on the timing behavior of the network. The main concern in real-time networks is to ensure that messages are delivered within their time constraints, i.e., their deadlines [15].
- **Modularity:** Elements used throughout the network should be easily integrated and interoperable with other elements. Thus, configuration and reconfiguration efforts should be limited and not require a complete redesign of the network [16], [17].
- **Fairness:** High-priority traffic should not impact the performance of lower-priority traffic excessively, considering this traffic still has real-time requirements in some cases [16].

[1]The definition used in this dissertation comes from the common use of the term *determinism* in the real-time networks community (See [9], [20]), which is similar to the definition of *timing predictability* in the rest of real-time research domain (See [21])

- **Complexity:** The design and implementation of the network should not be overly complex, making it difficult to manage and maintain. This includes considerations for network configuration, monitoring, and troubleshooting [18].

- **Costs:** The overall costs of the network should be kept low, including initial deployment costs, maintenance costs, and upgrade costs [17].

Specifically-created network protocols prior to the adoption of Ethernet in real-time networks were designed to meet these requirements [16], [22], [23], [24]. However, legacy Ethernet networks were not designed with real-time requirements in mind[2]. Originally, Ethernet was developed as a best-effort networking solution, focusing on maximizing throughput and flexibility rather than providing deterministic performance guarantees [25]. This leads to the presence of mechanisms that hinder Ethernet's capability to meet real-time networking requirements.

Legacy Ethernet networks provide *modularity* features through the use of standardized interfaces, allowing for easy interoperability between different devices and vendors. Moreover, their systems are plug-and-play, as no configuration parameters are required to connect devices into the network. The goal was to provide a protocol with low *complexity*. An example of this is the integration of the autonegotiation feature which allows devices to automatically configure themselves to the highest common speed and duplex mode supported by both ends of the link [25]. Legacy Ethernet networks were thus designed to provide high modularity and low complexity.

To extend the *complexity* aspect, the adoption of new technologies in real-time networks rendered these heterogeneous, with different technologies and protocols coexisting in the same network [27]. If we take the example of in-car networks, various fieldbuses are used for different applications, such as CAN for powertrain control, LIN for body electronics, and FlexRay for safety-critical systems [27]. The integration of Ethernet as a backbone network reflects a strategic shift toward unifying communication technologies under a single, open standard.

Another benefit of this is the reduced *costs* as Ethernet is already a widely adopted technology, thus leveraging existing infrastructure and devices to reduce costs [17].

Nevertheless, neither Ethernet [25] nor Switched-Ethernet [26] provide mechanisms to ensure *reliability*, *determinism*, or *fairness*. Regarding *reliability*, there are no real mechanisms to ensure this. The physical layer of Ethernet is robust and provides physical mechanisms to ensure a reliable connection; nevertheless, there are no higher-layer mechanisms to ensure frame delivery outside of those proposed to deal with cyclic dependencies from Spanning Trees [26].

Concerning *determinism*, the Full-Duplex aspect of Ethernet eliminates collisions, which is a step towards determinism. Moreover, through the VLAN tag we can manage the Quality

[2]Note we call legacy Ethernet networks the version of Ethernet presented in IEEE 802.3-2002 [25] and IEEE 802.1Q-2005 [26]. This version is Full Duplex. Thus, we do not consider the issues previous half-duplex versions contained regarding collision detection and avoidance and how this may impact real-time networks

of Service (QoS) through assigning priorities to different traffic classes and allowing to privilege real-time traffic over best effort traffic throughout the network. However, the use of NP-SP scheduling at the output ports, defined in the bridge operation clause of IEEE std 802.1Q-2005 [26], introduces a *fairness* issue as medium and low-priority traffic with timing constraints can be unduly delayed by high-priority traffic bursts. It is crucial to distinguish here between strict priority enforcement and fairness; while NP-SP faithfully respects the priority hierarchy, it is inherently unfair for lower-priority streams that still possess hard real-time constraints. This mechanism can lead to starvation, where lower-priority real-time traffic effectively loses access to the transmission medium, failing to meet deadlines despite being valid, critical traffic.

Thus Legacy Ethernet, although of interest, as it provides modularity, low complexity and network speeds required by modern real-time applications [28], lacks main real-time features such as *reliability*, *determinism* and *fairness*. These lacking features appear in the AVB standard and are further extended in the TSN standards, which are today merged to the current IEEE 802.1Q standard [29]. This standard defines how network bridges (commonly known as switches) operate.

Figure 2.1 illustrates the different mechanisms that constitute TSN and their associated standards. We differentiate four main categories within the TSN family of standards: Time Synchronization, Traffic Shaping for Bounded Low Latency, Reliability and Dedicated Resources and API's for Resource Management. From this general overview, we can see that TSN aims to provide mechanisms that answer the stated real-time networking requirements. *Reliability* needs are answered through the reliability mechanisms presented in the figure, specially FRER [30], which adds redundancy to network flows and Per-Stream Filtering and Policing (PSFP) [31], which is aimed at traffic policing for flows entering a switch. *Determinism* is given through the traffic shaping for bounded low latency mechanisms, some of which are enabled by the use of time synchronization. In addition, a specific mechanism from the traffic shaping set is directly aimed at improving *fairness*, the CBS defined in IEEE 802.1Qav [32].

The *Modularity* and *Complexity* aspects are inherited from the classical Ethernet design principles. Moreover, resource management mechanisms may improve modularity and reduce complexity by providing standardized interfaces for network configuration and management. The addition of new mechanisms adds more complexity and is the subject of ongoing research. We will introduce this topic in the Section 2.3. We also specify how these mechanisms fulfill the real-time networking requirements in Section 2.3. Finally, the last layer of complexity comes from the transition itself from legacy Ethernet to networks implementing TSN mechanisms. Since Ethernet is already present in many real-time systems, the transition to TSN must be done in a way that allows coexistence with legacy Ethernet devices. This is the subject of ongoing research, and we will discuss it further in Section 2.4.

Finally, regarding *costs*, the adoption of TSN mechanisms can lead to increased costs due to the need for specific hardware and software to support these features. This contradicts the philosophy of massively adopting Ethernet in real-time networks, where the idea is to be able to leverage existing infrastructure and devices. However, as TSN becomes more

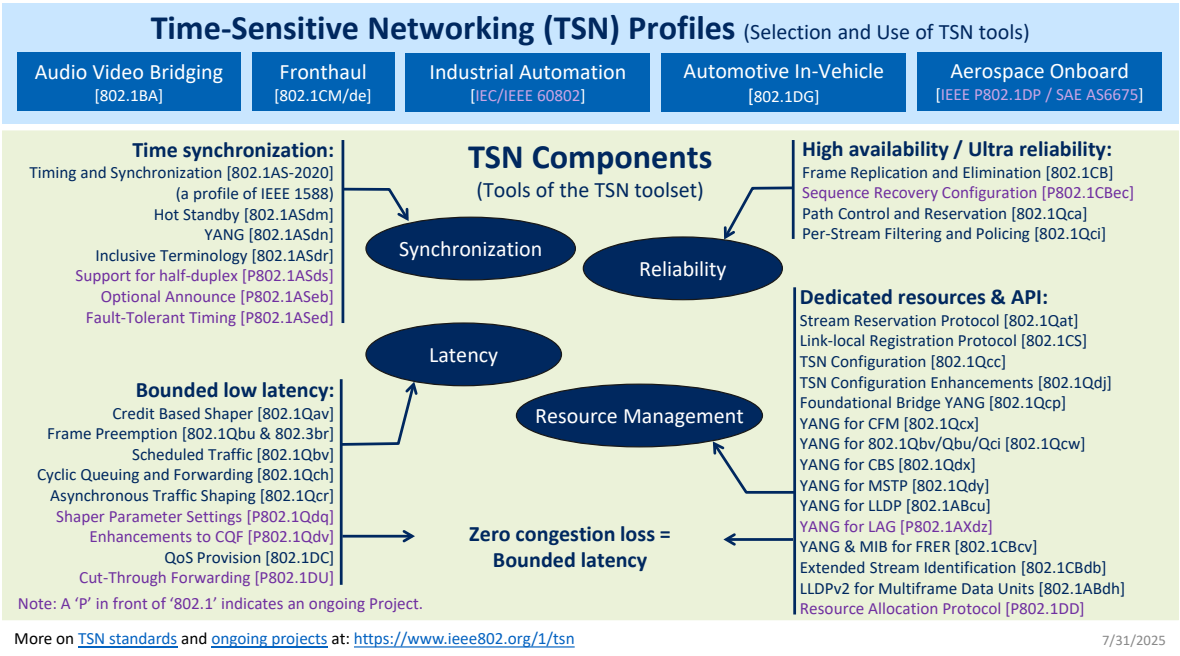


Figure 2.1: Summary of TSN mechanisms [9]

widely adopted, the costs associated with these mechanisms are expected to decrease, making them more accessible for real-time network deployments [17]. In the meantime, solutions are necessary to enable transition to TSN while keeping costs low.

2.3 TSN versus real-time networking requirements

2.3.1 Reliability

As stated, TSN provides mechanisms to ensure *reliability* in real-time networks. Two main domains are covered in terms of reliability: flow redundancy and policing. The former is addressed through the FRER mechanism defined in IEEE 802.1CB [30]. This mechanism allows the replication of frames at a source or relay node, sending them through disjoint paths in the network to then eliminate duplicates at a relay node or the final destination. The latter is addressed through the PSFP mechanism defined in IEEE 802.1Qci [31]. This mechanism allows switches to police incoming traffic streams based on predefined criteria, such as maximum frame size, bandwidth limits, and burst sizes. By enforcing these policies, PSFP helps prevent network congestion and ensures that high-priority traffic is not adversely affected by misbehaving or non-conforming streams.

2.3.2 Determinism

The technical foundation for TSN standards, which ensure predictable network behavior (i.e., *determinism*), is the IEEE 802.1Q standard [29]. The specific features that make up TSN are added to this main standard through various amendments.

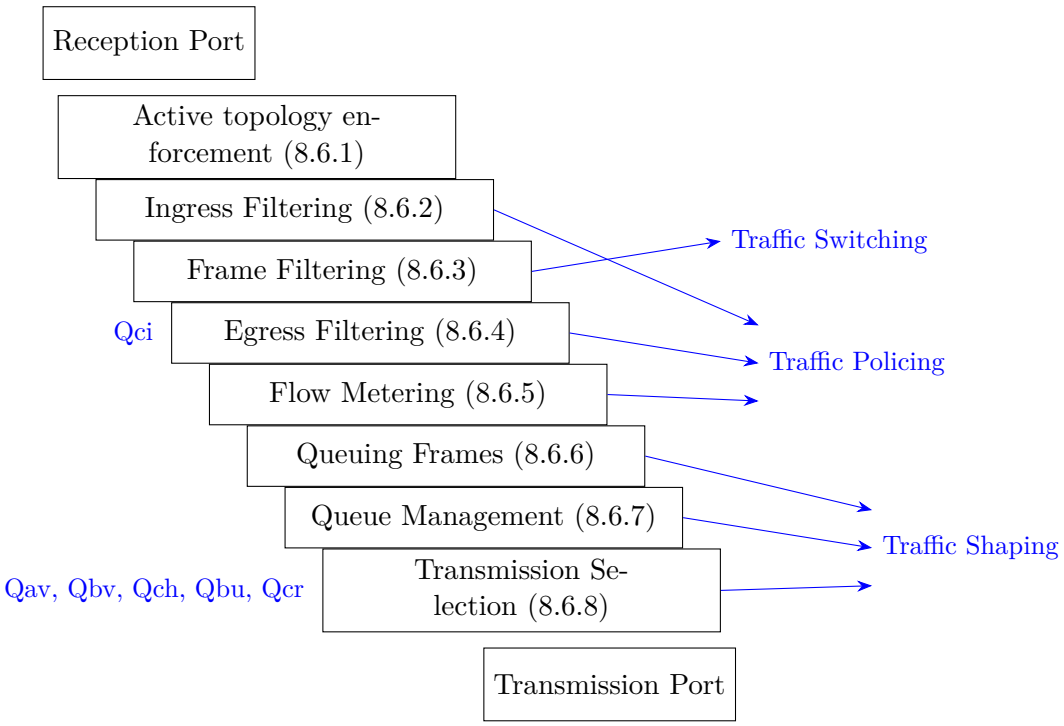


Figure 2.2: Bridge operation and TSN mechanisms (adapted from [29])

Clause 8 of the standard is particularly important, as it describes the forwarding functions of bridges. The main TSN-related amendments to this clause introduce several key mechanisms:

- **Asynchronous Traffic Shaping (ATS) (IEEE 802.1Qcr [33]):** Uses a per-stream, token-bucket-based algorithm to regulate traffic, providing maximum latency guarantees without requiring network-wide time synchronization.
- **TAS / Scheduled Traffic (IEEE 802.1Qbv [34]):** Applies a time-based schedule to control when specific traffic classes can transmit to guarantee bounded jitter and low latencies. This requires network-wide time synchronization to ensure that all devices adhere to the same schedule.
- **Cyclic Queuing and Forwarding (CQF) (IEEE 802.1Qch [35]):** Operates on a cyclic scheduling principle, using two queues per output port that alternate roles (storing or forwarding) based on a common time cycle. This mechanism is designed to provide guaranteed delay and limited jitter by storing frames received during one cycle and forwarding them during the next one.

- **Frame Preemption (IEEE 802.1Qbu [36]):** Allows higher-priority frames to interrupt the transmission of lower-priority frames at the Media Access Control (MAC) layer, which reduces latency for time-critical applications.
- **CBS (IEEE 802.1Qav [32]):** Uses a system of credits to regulate the transmission of certain traffic classes, reducing their burstiness, improving fairness for lower-priority traffic.

Figure 2.2 is a diagram that shows the general operation of a bridge and where these different TSN mechanisms fit into its design. In Clause 8, the IEEE 802.1Q standard also describes a queuing model for each transmission port. Each port can use up to eight separate traffic classes, and each class has its own First-in First-out (FIFO) priority queue. An incoming frame is assigned to one of these queues based on its priority level. This priority is indicated by the Priority Code Point (PCP) field, which is part of the frame's VLAN tag. The standard defines eight priority levels, 0 (lowest) to 7 (highest)[3], and each level is mapped to a specific traffic class [29, Annex I].

Once frames are in their queues, a scheduling algorithm decides which queue transmits next. The default mechanism is the NP-SP (Non-Preemptive Strict Priority) algorithm. This means the scheduler always selects the highest-priority queue that has a frame ready to send. Non-preemptive means that if a lower-priority frame is already being sent, its transmission must finish; it cannot be interrupted by a new, higher-priority frame. TSN improves this basic model by adding other transmission selection methods (like CBS, CQF, scheduled traffic, ATS, etc.) that work together with NP-SP. Therefore, for a frame to be sent when these TSN mechanisms are active, two conditions must be true:

- First, the specific transmission algorithm controlling that frame's queue (e.g., CBS, CQF, Scheduled traffic, ATS, etc.) must allow it to be sent.
- Second, the frame's queue must have the highest priority among all queues that are also ready to send.

These additions to the original NP-SP scheduler help ensure that data is handled with the correct priority to meet its given deadlines.

2.3.3 Modularity

Each output port of a TSN bridge consists of up to eight FIFO queues, each associated with a priority level. Incoming frames are placed in the queue corresponding to their priority. The transmission of each queue is governed by its Transmission Selection Algorithm (TSA) and an optional TT gate. The TSN standards allow for different TSAs to be implemented for each queue, providing flexibility in how traffic is managed. For example, one queue might

[3]Although in this dissertation, we will follow the inverse order to adopt a more intuitive notation, aligning priority escalation directly with numerical incrementation.

use the CBS mechanism, while another might use ATS. Other queues may use other TSAs. This modular approach allows network designers to tailor the behavior of each queue to the specific requirements of the traffic it handles.

Moreover, the use of standardized protocols and interfaces in TSN ensures that devices from different manufacturers can interoperate seamlessly. This interoperability is crucial for building complex networks that integrate various components while maintaining real-time performance. We observe this through the effort of standardizing TSN configuration and management through a YANG model [37] through the various resource management standards presented in Figure 2.1.

Moreover, as being an extension of the Ethernet protocol, TSN is interoperable with other real-time network technologies. As the need for Ethernet-based technologies grew, the need for interoperability between already established bus-based networks and Ethernet-based networks appeared. This led to the development of gateways and bridges that can connect different network types, allowing for seamless communication between devices on different networks. For example, in automotive systems, gateways are used to connect bus-based networks to Ethernet-based networks [28], [38]. These gateways translate messages between the different protocols, ensuring that data can flow smoothly across the entire system. This interoperability is crucial for integrating new Ethernet-based technologies into existing systems without requiring a complete overhaul of the network infrastructure.

2.3.4 Complexity

The design and implementation of TSN introduce significant additional complexity compared to traditional Ethernet. This complexity is due primarily to the variety of traffic-shaping mechanisms introduced to manage contention and the stringent requirement for network-wide time synchronization in TT solutions. Each mechanism needs careful configuration and management to satisfy *determinism* constraints, as this process involves tuning parameters and modeling intricate traffic interactions.

A major source of this complexity is the dependency on the synchronized clocks defined in IEEE std 802.1AS [39]. Mechanisms such as the TAS and CQF rely fundamentally on a synchronized network schedule. This adds a layer of architectural complexity where multiple synchronization parameters must be considered in the network's design [18]. Consequently, the state of the art in TSN research is heavily saturated with works addressing the configuration of synchronous mechanisms. Specifically for TAS, the calculation of the Gate Control List (GCL) represents a scheduling problem where frames must be placed in specific time windows without overlapping. This is mathematically formalized as a bin-packing problem, which is proven to be NP-complete [19]. As the network size or the number of critical streams increases, the computation time required to find a valid schedule grows exponentially. Current literature attempts to solve this using Integer Linear Programming (ILP) [40], Satisfiability Modulo Theories (SMT) [41], [42], [43], or meta-heuristics [44], [45], [46], yet these approaches often remain computationally expensive and inflexible to network changes.

Alternatively, ATS removes the dependency on global time synchronization, theoretically reducing the operational and architectural complexity. However, ATS shifts the complexity from the scheduling of data to the switch implementation itself. It requires complex internal operations based on the Urgency-Based Scheduler (UBS) [47] and per-stream state management. This increases the complexity of the switch silicon, meaning ATS support is typically found only in newer, more advanced hardware generations, potentially limiting its immediate availability in cost-sensitive or legacy-constrained deployments.

In contrast, CBS presents an appealing compromise. Like ATS, it operates without the need for global synchronization, but it employs a much simpler queuing logic based on credit accumulation and consumption. This simplicity has facilitated its widespread adoption, even in legacy AVB-capable switches. Despite this lower implementation complexity, CBS is not devoid of configuration challenges. The core difficulty lies in defining the appropriate `idleSlope` (bandwidth reservation) for each class of traffic. While the basic operation of CBS is well-understood, configuring the `idleSlope` to guarantee worst-case latency bounds without causing excessive bandwidth waste or impacting lower-priority traffic remains a non-trivial problem. Standard approaches often result in pessimistic over-provisioning [32], [48]. More research has been done on this subject leveraging holistic schedulability analysis techniques to obtain the minimum IdleSlope needed for a scenario with a single switch [49], [50], moreover other, better adapted frameworks such as NC [51] are used in the literature [52], [53] although they remain inefficient [54]. Authors in [54] provide an improved methodology that is faster, but does not ensure schedulability for all shaped flows. Consequently, a gap remains in the state of the art regarding efficient, automated methods for determining optimal CBS parameters that balance latency requirements with network resource efficiency.

2.3.5 Fairness

The CBS mechanism defined in IEEE 802.1Qav [32] specifically addresses the *fairness* requirement in TSN networks. CBS is designed to manage the transmission of high-priority traffic classes, to mitigate their burstiness. By implementing a system of credits, CBS regulates when frames from shaped traffic classes can be transmitted. This regulation helps to ensure that lower-priority traffic is not unduly delayed by bursts of higher-priority traffic, thereby improving overall network fairness and performance. This behavior improves fairness compared to the default NP-SP scheduling mechanism, where high-priority traffic can monopolize the transmission medium, leading to significant delays for lower-priority traffic. By controlling the transmission of high-priority traffic, CBS helps to balance the needs of different traffic classes, ensuring that all traffic types receive appropriate access to network resources.

2.3.6 Costs

The implementation of TSN mechanisms can lead to increased costs due to the need for specialized hardware and software to support these features. This includes the cost of upgrading

existing network infrastructure to be TSN-capable, as well as the ongoing costs associated with maintaining and managing a TSN network. However, as TSN becomes more widely adopted, the costs associated with these mechanisms are expected to decrease, making them more accessible for real-time network deployments [17]. In the meantime, solutions are necessary to enable transition to TSN while keeping costs low.

Various studies have focused mainly on the performance and reliability benefits of TSN, while overlooking the economic implications of its deployment. Works considering TAS and CBS focus on a *full deployment*, considering the use of both mechanisms throughout the entire network, some examples of these works are [19], [41], [44], [52], [53], [55]. However, a full deployment may not be economically feasible for many organizations, especially those with existing legacy Ethernet infrastructure. Therefore, there is a need for research into partial deployment strategies that can leverage the benefits of TSN while minimizing costs. Chaine et al. [56] provide a partial deployment strategy for TAS, where only the last-hop output ports implement TAS. Moreover, Thomas et al. [57] study the use of ATS to cut cyclic dependencies thus providing another application for partial deployment of TSN mechanisms to use them only when needed. However, to the best of our knowledge, no studies have been conducted on the partial deployment of other TSN mechanisms. This represents a significant gap in the literature, as partial deployment strategies could provide a more cost-effective way to implement TSN in real-world networks.

2.3.7 Discussion

The analysis of real-time network needs reveals a clear trade-off between performance and complexity. We observe that in TSN, the *determinism*, *reliability*, *complexity*, *modularity*, and *fairness* are not complementary. This is most evident in the suite of TSN shaping mechanisms, where achieving the highest levels of determinism often incurs prohibitive complexity.

The synchronous, TT mechanisms, namely TAS and CQF, exemplify this challenge. While they offer the strongest deterministic guarantees by creating isolated communication slots, they fail significantly on the metric of complexity. Their reliance on network-wide, high-precision time synchronization introduces significant architectural and maintenance overhead. More critically, the computational complexity of TAS scheduling, being NP-Hard, presents a fundamental barrier to scalability and modularity. Any change in network topology or stream requirements necessitates a complete, and costly, recalculation of the entire network schedule, making such solutions rigid and impractical for dynamic environments.

Asynchronous mechanisms like ATS and CBS were developed to address these complexity concerns. ATS successfully removes the need for global synchronization, solving the architectural overhead issue. However, it embeds the complexity into the switch fabric itself. Its intricate per-stream state management requires advanced hardware support, which can limit interoperability with older equipment and increase initial deployment costs. In addition, ATS demands heavy configuration at each switch where it is implemented, as time differences between the clocks used for ATS operation may cause instability in the network if not

considered [58].

This leaves CBS as a highly pragmatic candidate for balancing real-time needs with practical deployability, particularly in networks that must integrate with legacy devices. CBS successfully provides bounded latency (determinism) and prioritized access (reliability and fairness) without a need for a global time synchronization or specialized, next-generation switch silicon. Its asynchronous, self-contained operation makes it inherently modular and scalable. However, as established, the challenge of CBS is not one of operational or hardware complexity, but one of *configuration*. Moreover, current works assume CBS, when used, is used throughout the path of the real-time flows affected to the CBS class, which goes against the modularity aspect required by real-time networks. The state of the art has demonstrated the operational viability of CBS, but a significant research gap exists in how to optimally configure its parameters, specifically the `idleSlope` and how to deploy CBS to ensure interoperability with legacy Ethernet devices in the network. Inefficient configuration leads to either pessimistic bandwidth over-provisioning, which violates fairness and wastes resources, or insufficient allocation, which breaks deterministic guarantees. Therefore, research into methodologies for the correct and efficient deployment and configuration of CBS is paramount for the practical deployment of next-generation real-time networks. Finally, partial deployment strategies for CBS remain unexplored, representing a critical area for future investigation to enable cost-effective TSN adoption, increasing the need to deepen the understanding on CBS use.

This dissertation aims to address this gap by answering the following research questions:

- R.Q.1: *Where: Where can CBS be effectively deployed to leverage its benefits, while maintaining interoperability with legacy Ethernet devices?*
- R.Q.2: *When: When is CBS a good solution to improve network performance, considering the different traffic patterns and network topologies?*
- R.Q.3: *How: How can CBS parameters, particularly IdleSlope, be optimally tuned to balance deterministic guarantees with efficient resource utilization?*

2.4 CBS and State of The Art

In this section, the CBS mechanism defined in IEEE 802.1Qav [32] is detailed. Afterwards, an overview of current developments in the literature regarding CBS and particularly the existing performance analyses, the approaches for its configuration and placement in TSN networks as well as its interaction with other TSN mechanisms is developed.

2.4.1 CBS Algorithm

The CBS mechanism operates at the output ports of TSN switches. Each output port consists of up to eight FIFO queues, each associated with a priority level. Incoming frames are placed in the queue corresponding to their priority. The transmission of each queue is governed by its TSA and an optional TT gate, which we do not consider in this work. For CBS, the TSA is the CBS itself.

The queues implementing CBS operate under a credit value that determines whether frames can be transmitted. Frames are only allowed for transmission if the credit of their queue is non-negative. Moreover, the credit of a queue increases with the rate called *IdleSlope* when frames to be scheduled cannot be sent due to an ongoing transmission of other queues or because the credit is negative. Conversely, the credit decreases with the rate called *SendSlope* during frame transmission.

To explain the operating principle of the CBS we define a set of rules to set how frames are scheduled. These rules help understand how the interaction between class priorities, available credit, and queue status control the transmission process defined for CBS.

1. **Transmission Line Availability:** When the transmission line is unoccupied, the CBS selects a frame for transmission if two conditions are satisfied:
   (a) The queue contains data ready for transmission.
   (b) The class has non-negative credit.
2. **SendSlope Credit Reduction:** During transmission, its credit is linearly decreased at the rate of SendSlope.
3. **IdleSlope Credit Increment:** Credit increases at the rate of IdleSlope under the following circumstances:
   (a) The queue holds data.
   (b) Other classes are in the process of transmitting.
4. **Credit Reset:** Positive credit linked to an empty queue resets to zero.
5. **Negative Credit Correction:** If credit were to be negative when a frame transmission completes, it is increased at rate IdleSlope until it reaches zero.

Following these rules, the credit evolution for a queue can be described through a series of phases, as illustrated in Figure 2.3. To illustrate this, we consider an output port implementing CBS for priority queue $p = 0$. Consider at $t = 0$ two queues have available traffic to schedule: priority $p = 0$ and priority $p = 1$. Additionally, there is a best-effort (BE) frame being transmitted. The credit evolution for the queue with priority $p = 0$ can be described through the following phases:

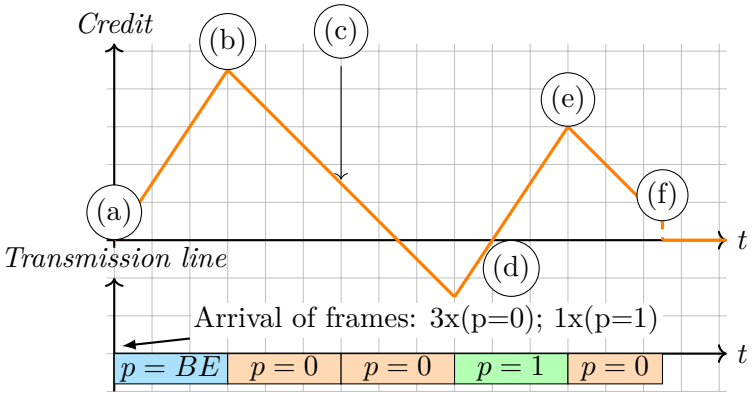


Figure 2.3: Credit evolution of a CBS queue over time

- **Phase (a):** At time $t = 0$, three frames are waiting for transmission in priority 0, while one frame is queued in priority 1. One frame is already being transmitted from the BE queue. Therefore, the credit for queue $p = 0$ increases with the *IdleSlope*.
- **Phase (b):** After the BE frame has finished transmission, both the queue with priority $p = 0$ and the queue with priority $p = 1$ have eligible frames and the queue with priority $p = 0$ has positive credits. So the priority is used to decide on transmission. Priority 0 sends first.
- **Phase (c):** Priority 0 frames are eligible for transmission as long as at the beginning of the transmission, the credit of this queue is non-negative. The credit of this queue decreases with the *SendSlope*[4] during the transmission of frames corresponding to priority 0.
- **Phase (d):** After having scheduled two frames of priority 0, the credit of this queue is negative, thus priority 1 is allowed to send despite having one frame of priority 0 left to transmit. During this time, the credit is replenished for priority 0 at rate *IdleSlope*.
- **Phase (e):** After finishing sending the frame of priority 1, the credit of priority 0 is greater to 0, therefore, the last unsent frame may be transmitted.
- **Phase (f):** If a queue is empty, but the credit is positive, the credit of this queue is set to zero.

2.4.2 State of the Art on CBS

2.4.2.1 Performance Analysis Approaches

In the context of real-time networks, timing analysis including the possibility to study delay bounds of traffic traversing a network is of crucial importance, as it allows to provide formal

[4]SendSlope is obtained by removing IdleSlope from the line rate

proof that flows transmissions will stay within their assigned deadline. Through this intent, CBS has been thoroughly studied in the literature, with various methods proposed to formally analyze its performance in TSN networks. The methodologies can be divided into four main categories: Holistic schedulability analysis, Trajectory approach analysis, Compositional Performance Analysis (CPA) and NC-based approaches, and other approaches that cannot be categorized into a classic type of analysis.

Holistic schedulability analysis approaches The origin of holistic approaches comes from the field of real-time scheduling, where the main objective is to determine whether a set of tasks can meet their deadlines when scheduled on a processor [59]. The concept of a holistic analysis for networks was first introduced in [60], where the authors applied the holistic approach to analyze the worst-case delay of messages in Controller Area Networks (CAN). The main idea is to consider all of the elements that might cause interference to obtain the worst-case traversal time between a sending application and a receiving application. These concepts have mainly been applied to CBS performance analysis through a concept known as Busy Period Analysis (BPA)[61]. This method involves identifying periods during which the network is continuously busy transmitting frames, and analyzing the worst-case delay that a frame might experience during these periods. The analysis considers the interactions between different traffic classes, their priorities, and the CBS parameters to compute the worst-case delay bounds.

The first authors to apply BPA to CBS were [62], who proposed a method to compute delay bounds for a frame subject to CBS scheduling by considering the busy periods generated by credit recovery and higher priority traffic under NP-SP scheduling. Subsequent works have built upon this foundation, first, extending the analysis to networks implementing scheduled traffic mechanisms [49]. Then, an extension of BPA was proposed in [63], named Eligible Intervals. The extension was meant to improve the tightness of the current BPA for CBS by considering only the intervals of time where CBS had traffic to schedule and was able to. The main limitation of these works is their applicability to single-hop networks. Furthermore, holistic approaches struggle to address the compositional nature of large, multi-vendor networks, where the internal details of every switch may not be fully exposed or identical, limiting their modularity.

More recently, Bujosa et al. [64] have proposed a node-level analysis through the Eligible Interval approach of CBS to obtain end-to-end delay bounds by aggregating the delays at each node along a flow's path to improve delays tightness. Their results are compared to NC results based on the Total Flow Analysis (TFA) [65] algorithm, which is known as a simple approach and not tight compared to newer NC results [66], [67].

Trajectory approaches Compared to holistic approaches, trajectory approaches focus on analyzing the complete path of a frame through the network, rather than just individual nodes [68]. The main idea is to track the trajectory of a frame as it traverses the network, considering the effects of scheduling and queuing at each node along its path. This method

allows for a more detailed analysis of the interactions between different traffic flows and the impact of network mechanisms on the end-to-end delay, but is difficult to scale to large networks with multiple data flows traversing the network. The first work to apply trajectory analysis to CBS was by Li & George [69]. In this work, the authors analyzed delay by extending the Trajectory Approach to specifically model AVB networks. Their method calculates the worst-case end-to-end delay by summing all interference from higher and same-priority flows, non-preemptive blocking from lower-priority flows, and switching latencies. This model integrates the unique delays caused by CBS through computing the credit replenishment time, and is refined to be more accurate by accounting for factors like frame serialization and overlapping delays from queuing.

CPA CPA is a modular approach [70], where the idea is to compute local interference for all the scheduling nodes and then applying this iteratively to every flow in the network until either formally proving every component is schedulable or violates their deadline. Diemer et al. [48] apply CPA to the AVB context by proposing model of delay of each individual node implementing CBS. Axer et al. [71] extend the work of [48] as they show their approach presented an optimistic analysis of CBS.

NC-based approaches NC [51] is a theoretical framework, first appeared in [72], which is used to analyze the performance of communication networks, particularly for providing deterministic QoS guarantees. It models data flows using arrival curves (e.g., $\alpha(t)$), which define the maximum amount of data that can enter the network over any time interval, and models network elements (like switche's schedulers) using service curves (e.g., $\beta(t)$), which describe the minimum processing capacity or bandwidth guaranteed by that element. By applying mathematical operations based on min-plus algebra [73], this framework allows to calculate strict, worst-case bounds, such as the maximum delay (latency) a frame will experience and the maximum backlog (buffer size) required to prevent data loss. Several works have applied NC to analyze CBS performance in TSN networks. The first author to do so was Queck [74], to which, for a NP-SP scheduler utilizing CBS at priorities 1 and 2, he was able to derive the service curve of each CBS instance. Later, the work of Ruiz de Azua & Boyer [75] reveal the proofs for the derived service curves had weaknesses although correct. Their work provides a more robust derivation of the service curves for the specific case of two CBS instances at the second and third queues of highest priority. Following these works, Mohammadpour et al. [76] and Zhao et al. [77] provide a more general derivation of the service curve for CBS at any priority level. In addition, Reinmann et al. [78] apply the Modular Performance Analysis (MPA) framework [79] by using real-time calculus [80], a fork of NC [51], to model local scheduling delay caused by each CBS instance and then aggregate these delays to obtain formal delay bounds for flows from ECUs traversing E/E vehicle networks.

Other approaches Bennamar et al. [81] have proposed an approach they named *Forward End-to-end Delay Analysis (FDA)* to analyze end-to-end delay bounds in TSN networks im-

plementing CBS. Their approach combines the node-level analysis of CBS, which consists on formally finding the maximum interference from all other competing frames to then find the end-to-end delay computation through the FDA framework by aggregating the delays at each node along a flow's path.

Discussion From the presented state of the art on formal analysis frameworks for CBS, we observe that multiple methods have been proposed to analyze CBS performance in TSN networks. One stands out from the others as it provides computational modularity and scalability while providing end-to-end formal delay bounds: NC-based approaches. This is due to the fact that NC provides a mathematical framework to model and analyze the performance of network elements in a modular way, that use generic functions to compute tight delay bounds [66], [67]. Each network element, can be modeled independently using service curves, which describe the minimum service guaranteed by that element. Unlike holistic approaches that require a complete system view for every recalculation, NC allows for component-based analysis, essential for heterogeneous networks where legacy and TSN devices coexist. CBS has been thoroughly studied in the literature through NC, with multiple works deriving service curves for CBS at different priority level. This provides a solid foundation for analyzing CBS performance in TSN networks using NC, which is why we use it as our tool of choice for analyzing CBS performance in this dissertation.

2.4.2.2 CBS Deployment and Configuration

Works that focus on CBS use, typically consider *full CBS deployment*. Throughout this document, we refer to full CBS deployment when CBS is considered for all output ports of the network switches. We observe this trend in different works using different approaches to configure CBS, such as: [48], [49], [52], [53], [54], [64], [82], [83], [84]. This ubiquitous assumption in the literature presents a practical limitation, as a full CBS deployment is expensive to implement and maintain across an entire network.

CBS configuration has been mainly studied to obtain an IdleSlope value that would allow the schedulability of the shaped priority-queue. According to IEEE Std 802.1Qav [32], the IdleSlope should be set as the bandwidth requirement of the class subject to CBS. Diemer et al. [48] argue that this method of bandwidth reservation may lead to unscheduled shaped flows. This behavior is also observed by Axer et al. [71], where they set their IdleSlope values to double the bandwidth requirement. To generalize, Diemer et al. propose an over-reservation approach, which involves multiplying the IdleSlope specified in [85] by an arbitrary factor. They also highlight the potential blocking effect of CBS, which can significantly increase latency for prioritized traffic, emphasizing the role of IdleSlope selection. Ashjaei et al. [49] compute an IdleSlope that ensures the schedulability of shaped traffic, integrated with scheduled traffic mechanisms from [29]. They do so by computing a local deadline to which they compare the local delay their approach is able to compute. Through this inequality, they are able to find an appropriate IdleSlope. Cao et al. [82] propose an approach to compute bandwidth reservation which is independent from interference caused by other

traffic classes. In addition, Kong et al. [83] perform yet another busy period analysis to obtain the maximum delay for the frames of a flow caused by queuing and the subsequent transmission link. Through their model, they are able to find an upper bound for this delay which in turn they use to express a value for the IdleSlope. Finally, Houtan et al. [84], use the approach presented in [49] to provide an IdleSlope for an output port using CBS in addition to scheduled traffic. They specifically analyze situations where CBS instances are used in the second and third most prioritized queues. They propose an IdleSlope based on the credit needed to make all flows shaped to be schedulable. The drawback of these approaches is that they are meant for single switch scenarios.

There are also simulation driven approaches to configure CBS. For example, Nitta et al. [86], who propose a solution based on simulation models, where they introduce a constraint-based linear optimization model to compute suitable IdleSlope values based on end-to-end delays for shaped flows. Finally, to derive IdleSlopes for CBS deployed in complex networks with multiple switches through formal analysis, other recent studies use NC. Maile et al. [52] and Zhao et al. [53], for example, propose approaches based on CBS NC models. Both approaches use a method that derives the necessary Service Curve to schedule network flows and, from this Service Curve, compute an appropriate IdleSlope for configuring CBS. These are developed with efficiency in mind as they are meant for scenarios such as online CBS configuration. Maile et al. adjust flow paths if resources are insufficient for reservation, while Zhao et al. compute local delays using the Service Curves per output port[cite: 223, 224]. Finally, Yu et al. [54] extend the work in [53] by proposing a methodology for online configuration in ATS/CBS deployments, which shows much more efficient results than the ones proposed by Zhao et al., which should be used for offline reconfiguration. These advances show the ability of NC for analyzing end-to-end performance in complex topologies, validating our choice of using it in this dissertation.

In addition, some works have taken an interest in credit behavior in CBS when paired to scheduled traffic at the output port level. Boyer & Daigmorte [87] have taken an interest into the impact scheduled traffic can have over CBS credit behavior. The amendment IEEE 802.1Qbv [34] (which defines TAS), extends the IEEE 802.1Q standard to include a gating mechanism that opens gates that allow a priority queue to be scheduled based on a time schedule. With this amendment, CBS behavior is modified. This is specified through Note 1 of Clause 8.6.8.2 of [29], where it is stated that when the gate of a CBS-shaped priority queue is closed, the credit increase is paused. This behavior is known as frozen credit. Boyer & Daigmorte study this impact, specifically during the time period just before the gate for CBS-shaped traffic close. They discuss whether the credit has to be frozen or not during this time period, as assumed by most research papers at that time. They conclude that freezing the credit in moments prior to gate closing can lead to worse delay bounds for CBS-shaped traffic, whereas just freezing credit when the gate is closed leads to better delay bounds. The rest of the papers focus on finding an appropriate IdleSlope value for shaped traffic, whether with or without scheduled traffic.

Discussion Current literature assumes a costly full CBS deployment, as now every switch must support CBS. An alternative to this is what we call *partial CBS deployment*, where CBS is only deployed at specific points in the network. This approach allows for a more targeted use of CBS, focusing on areas where its benefits are most needed, while avoiding unnecessary complexity and resource consumption in parts of the network where CBS may not be required. Most existing literature inherently assumes a full CBS deployment scenario, where the network is built entirely from scratch with the latest TSN-capable hardware. This overlooks the practical reality of partial CBS deployments, where TSN must be integrated into existing legacy infrastructures. This approach has not been studied in the literature and is a clear research gap that this dissertation aims to address. The goal is to support legacy Ethernet devices while ensuring the schedulability of both shaped and non-shaped traffic, thereby facilitating a more targeted and efficient use of CBS resources rather than unnecessary network-wide implementation.

In addition to these practical considerations, there are also quantitative-oriented reasons to consider Partial CBS Deployment. It is a well-known fact that CBS introduces a *blocking effect* on high-priority shaped traffic [48]. This effect can lead to increased latency for high-priority traffic, which is counterproductive in real-time networks. This blocking comes from the credit behavior of CBS, once spent (i.e. when credits are negative), the shaped queue cannot schedule frames until credit has been recovered. Thus, if contending traffic of lower priority is waiting, it will be scheduled, thus delaying the shaped class that still has frames for transmission. By strategically deploying CBS only where it is most needed, we can potentially reduce this blocking effect as less CBS instances are used, thus limiting the potential points where blocking effect occurs, extending the interest of performing Partial CBS Deployment.

By leveraging the NC framework, this dissertation proposes an approach to CBS configuration that balances performance guarantees with these practical deployment considerations. The resulting framework allows network designers to make informed decisions about CBS placement.

2.4.2.3 Interaction of CBS with Other TSN Mechanisms

Since CBS's first appearance was on the AVB set of standards, multiple studies have considered CBS in conjunction with TAS. A simulation study to understand the coexistance of TAS and CBS is given by Meyer et al. [88], then, a NC model that combines both TAS and CBS is presented by Zhao et al. [89]. This analysis allows analysis for an arbitrary number of classes using CBS, considering both standard credit behavior and more generally assumed credit behavior but deviating from the standard 802.1Q. Other works consider this combination in their timing analysis, such as Ashjaei et al. [49], who provide a holistic schedulability analysis for networks implementing both TAS and CBS. Moreover, Boyer et al. studied the impact of the frozen credit policy introduced by the standard caused directly by the combination of TAS and CBS. The main concern in these studies regarding CBS is how the credit behavior is affected by the gating mechanism of TAS. The main conclusion is that the solution proposed by the standard of freezing credits is sound and should be kept this way [87], [89].

CBS has also been studied in combination with ATS, where ATS instances feed CBS-shaped queues. Mohammadpour et al. [90] propose such a scheme with the intent of reducing inter-class and intra-class burstiness at the output port level. Zhao et al. [91] provide a NC model for ATS+CBS deployments, where they provide service curves for ports using CBS and ATS at different port priority levels, the main challenge here is providing a residual service curve for this combination.

Discussion From the state of the art, we observe that CBS has been mainly studied in combination with other traffic shapers. Other TSN mechanisms have not been considered. The state of the art on other TSN mechanisms indicates a potential interest in further developing the understanding on how other mechanisms impact the use of CBS. Mainly, Thomas et al. [92] provide a NC-based analysis for FRER. Through their study, they conclude that flow elimination, a key part of the FRER mechanism, increases the burstiness at its output. This burstiness creates a specific, unaddressed need for traffic shaping immediately following FRER elimination points to restore traffic characteristics before it traverses the rest of the network. Therefore indicating a clear need for CBS and a better understanding on how these mechanisms interact. We note that FRER in general has not been studied in combination with CBS. Interactions between FRER and other TSN scheduling mechanisms, such as ATS [92], [93] and CQF [94], have been investigated, showing the need to consider FRER's effects carefully when configuring them. However, existing work on CBS deployment does not consider the impact of FRER, and FRER studies generally do not address CBS deployment. Thus, we identify a clear gap in the literature regarding the combined use of CBS and FRER in TSN networks. This dissertation aims to fill this gap by exploring how CBS can be effectively deployed and configured in networks implementing FRER, ensuring that both mechanisms can coexist while meeting the performance requirements of time-sensitive applications.

2.5 Conclusion

In this chapter we have discussed the requirements for modern real-time networks, highlighting the trade-offs between determinism, reliability, complexity, and modularity. Our analysis of the TSN standards reveals that while synchronous mechanisms like TAS and CQF offer strong guarantees, they do so at the cost of high architectural and configuration complexity. Asynchronous mechanisms, in contrast, offer a more flexible path forward.

Our interest in the CBS stems from its unique position as widely-available and asynchronous mechanism that balances performance with implementation simplicity. Unlike ATS, it does not require specialized next-generation hardware, and unlike TAS, it avoids the complexity of global time synchronization use. It has thus emerged as the most promising candidate for delivering deterministic guarantees in scalable and modular networks, particularly those that must integrate with existing infrastructure.

However, our survey of the state of the art has identified two critical and unaddressed gaps in the literature. First, existing research almost universally assumes a *full CBS deployment*,

where every switch in the network is CBS-capable. This assumption overlooks the practical reality of heterogeneous environments, where new TSN components must interoperate with legacy Ethernet devices. Consequently, there is no established methodology for a *partial CBS deployment*, leaving a significant gap in modular and cost-effective network design.

Second, we have identified a critical lack of research into the interaction between CBS and other TSN mechanisms, specifically FRER. The literature indicates that the FRER elimination phase can itself introduce significant traffic burstiness—the very problem CBS is designed to mitigate. The challenge of how to configure CBS to manage FRER-induced burstiness has not been addressed.

These identified gaps directly map to the research questions of this dissertation. By exploring a *partial CBS deployment* approach, we can directly answer **R.Q.1** (*Where* to deploy CBS) and **R.Q.2** (*When* it is an effective solution, such as in mixed-device networks). By analyzing the specific interaction with FRER, we can provide a deeper answer to **R.Q.3** (*How* to configure CBS) in the context of other TSN mechanisms to balance determinism and resource efficiency.

The following chapters of this dissertation will aim to fill this gap. We will propose a partial CBS deployment methodology based on the NC tool, providing the approaches to strategically deploy and configure CBS to ensure end-to-end performance requirements in time-sensitive networks.

Chapter 3

A Cost-Efficient Partial CBS Deployment Framework

Contents

3.1 System Model & CBS background 29
3.1.1 Network Model and Architecture 29
3.1.2 Traffic Model 29
3.1.3 A NC model for CBS 30
3.2 Motivation for Partial CBS Deployment 31
3.2.1 Partial CBS Deployment is cost-effective and Lowers Design and Operation Complexity 32
3.2.2 Delay improvement of high-priority shaped traffic through Partial CBS Deployment 32
3.2.3 Challenges and Considerations for a Partial CBS Deployment Strategy . . 33
3.3 The Partial CBS Deployment as an Optimization Problem 36
3.3.1 Problem Formulation 36
3.3.2 Problem Linearization and Relaxation 40
3.4 Framework for Partial CBS Deployment 44
3.4.1 Framework Overview for Optimizing Partial CBS Deployment 44
3.4.2 Burstiness-aware CBS Placement 45
3.4.3 Burstiness-Aware CBS Configuration 46
3.4.4 Heuristic Algorithm 48
3.4.5 Complexity Analysis Computation 53
3.5 Framework Validation Over an Automotive Use-Case 55
3.5.1 Network Topology and Architectural Context 56
3.5.2 Traffic Characteristics and Interference Patterns 56
3.5.3 Evaluation Scenarios 57
3.5.4 Results and Analysis 58
3.6 Conclusions 60

In the previous chapter, the main requirements for real-time networks, such as reliability, determinism, and modularity, were discussed. TSN provides the necessary mechanisms to meet these requirements. Among the various TSN standards, CBS was identified as a

promising solution; unlike TAS, it offers a strong balance between performance and complexity without requiring global synchronization.

However, as highlighted in the state-of-the-art analysis, most existing research assumes a *Full CBS Deployment*, modeling a homogeneous network where every switch is TSN-capable. This assumption poses a significant challenge for real-world deployments, where cost constraints are critical. Practically, a transition-friendly architecture is necessary; such an architecture must integrate both new TSN switches and older *legacy* Ethernet switches that use standard NP-SP logic. Currently, there is no clear methodology for a *Partial CBS Deployment* that mixes these devices.

This chapter addresses this gap. We propose a strategy for the partial deployment of CBS to provide a framework to answer the research questions defined in this dissertation:

- **R.Q.1:** *Where* can CBS be effectively deployed?
- **R.Q.2:** *When* is CBS a good solution to improve network performance?
- **R.Q.3:** *How* can CBS be configured to obtain the best performance trade-off in a partial deployment scenario?

Our motivation for this approach is based on two main goals. First, from a cost and modularity perspective, the proposed framework enables network designers to utilize existing legacy infrastructure. Consequently, TSN capabilities are introduced where they are strictly necessary. Second, from a performance perspective, it is established that legacy switches can exacerbate traffic burstiness, which affects real-time guarantees. By strategically placing CBS instances, we aim to reduce this burstiness while ensuring fairness and determinism in the network.

To achieve this, we use the NC framework. As concluded in Section 2.4, NC serves as an optimal framework for this analysis, as it provides a highly customizable mathematical framework to provide formal performance guarantees.

This chapter is organized as follows:

- We provide the system model we will use throughout this dissertation in Section 3.1.
- We define the problem motivation in Section 3.2, explaining the challenges for partial CBS deployment.
- We provide a formal definition of the Partial CBS Deployment Problem in Section 3.3.
- We present the proposed framework to solve the Partial CBS Deployment Problem in Section 3.4.
- We provide a performance evaluation of the proposed framework over an industrial use-case in Section 3.5.
- We discuss the key findings, contributions and limitations of these in Section 3.6.

3.1 System Model & CBS background

To provide a comprehensive understanding of the partial CBS deployment problem, we first need to define the system model used in our analysis. This model includes the context on CBS operation, a formal definition of network topology, CBS use and the characteristics of the traffic traversing it. We finish this section by detailing the mathematical model from NC state of the art to analyze CBS networks.

3.1.1 Network Model and Architecture

At its core, a Time-Sensitive Network (TSN) is composed of two primary equipment types: *End-Systems* (ES) and *Switches* (SW). Together, these are referred to as devices and are grouped in the set DEV. The ES function as the sources and sinks of traffic flows, representing devices like sensors, actuators, or controllers, while the SW are responsible for forwarding this traffic between them. These components are interconnected via Full-Duplex links to form potentially complex topologies, with each link having a global data rate capacity, denoted as C. Communication is handled by Ethernet interfaces containing *Output Ports* (OP), which manage the transmission and reception of data packets.

To formally represent this physical structure, we model the network as a directed graph, $Net = (OP, E)$. In this graph, the set of vertices, OP, corresponds to every Output Port in the network, and the set of directed edges, E, represents the Full-Duplex links connecting them. This graph-based representation effectively captures the network's complete topology.

Delving deeper into the functionality of each Output Port, its architecture is based on a system of priority queues managed by a Non-Preemptive Static Priority (NP-SP) scheduler. Each OP supports up to eight priority queues, denoted as $P = \{0, 1, 2, 3, 4, 5, 6, 7\}$, where a lower number indicates a higher priority, with $p = 0$ as the highest priority. Building on this architecture, the CBS mechanism can be configured for any given queue; in our model, CBS acts as the only possible TSA. An overview of the output port we just described may be found on Figure 3.1. We use a binary variable, CBS_{op}^{p}, which is set to 1 if CBS is enabled for priority p on port op. When active, the shaped queue is assigned an IdleSlope attribute, I_{op}^{p}, which dictates its credit accumulation rate. While our approach considers only CBS in Switches, its use in End-Systems remains optional, as they are often legacy devices lacking support for complex shaping algorithms. Nevertheless, the model remains valid even if CBS is deployed on them.

3.1.2 Traffic Model

Flows are defined by a set of abstract flow specifications, $\mathcal{F}$. Each specification $f \in \mathcal{F}$ is a tuple $(r^f, b^f, L^f, D^f, p^f)$, representing its source rate, source burst, maximum frame size, end-to-end deadline, and priority. A flow is considered *shaped* if it traverses at least one output port where CBS is active for its priority level.

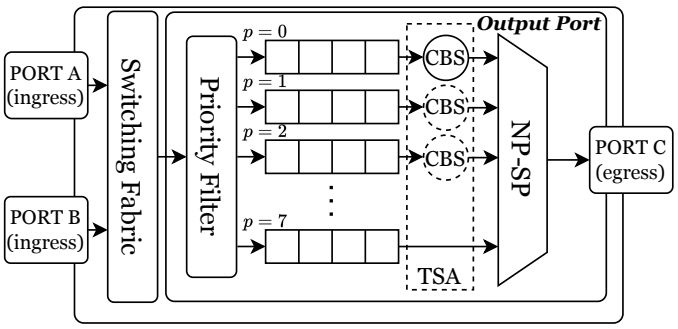


Figure 3.1: Detailed switch architecture, showing the ingress and egress pipeline of an Output Port.

Each flow f follows a pre-determined path $\phi^f = (op_1, op_2, \ldots, op_n)$, which is a sequence of Output Ports from its source to its destination; we assume these paths are known *a priori* and that flows do not change priority along their path.

To simplify analysis and notation, we define several subsets of flows. First, $\mathcal{F}(p)$ represents the set of all flows assigned to a specific priority p. Building on this, $\mathcal{F}(op, p)$ denotes the subset of flows with priority p that traverse a particular Output Port op. Finally, $\mathcal{F}(dev)$ is the set of all flows that pass through any port on a given device dev.

Table 3.1 summarizes the notations used throughout this chapter for clarity.

3.1.3 A NC model for CBS

To analyze the performance of CBS in our network model, we leverage the NC framework. NC is presented in Appendix A and generally provides a set of mathematical tools to model and analyze deterministic queuing systems. It does so by leveraging min-plus algebra on arrival bounds and minimal service curves to provide worst-case performance bounds such as end-to-end delay and backlog.

For CBS, the main NC results concerning CBS are in its Service Curve when integrated to an NP-SP scheduler. The Service Curve is given by Mohammadpour et al. and by Zhao et al. [76], [95][1] through Equation 3.1, which is represented as a Rate-Latency Service Curve[2].

$$\beta_{op}^{p}(t) = R_{op}^{p}\left[t - T_{op}^{p}\right] I_{op}^{p}\left[t - \frac{cred_{op}^{p,\max}}{I_{op}^{p}}\right]^{+} \tag{3.1}$$

[1] These two works provide the same Service Curve, but with different proofs.
[2] See Appendix A

Table 3.1: Summary of notations

General Network and CBS Notations	
OP, E	Set of output ports, Set of unidirectional links
op	An output port $\in OP$
C	Transmission capacity of a link
DEV	Set of devices ($SW \cup ES$)
SW, ES	Sets of switches and end systems
$op.dev$	Device associated with port op
$OP(dev)$	Set of output ports of device dev
P	Set of priorities $\{0, \ldots, 7\}$
CBS_{op}^{p}	Binary: CBS active at op for priority p
I_{op}^{p}	IdleSlope for CBS at op for priority p
$L_{op}^{>p,\max}$	Max. frame size of interfering traffic at op with lower priority than p
Flows Notations	
$\mathcal{F}$	Set of all abstract flow specifications
(r^f, b^f)	Source rate and burst size of f
L^f	Maximum frame size of f
D^f	End-to-end deadline of f
p^f	Assigned priority of f
ϕ^f	Path of flow f
d^f	End-to-end delay bound of f
$\mathcal{F}(p)$	Set of abstract flow specifications $f \in \mathcal{F}$ with priority p
$\mathcal{F}(op, p)$	Abstract flow specifications $f \in \mathcal{F}(p)$ traversing op
$\mathcal{F}(dev)$	Abstract flow specifications $f \in \mathcal{F}$ traversing dev

Where $cred_{op}^{p,\max}$ is the credit upper bound for priority p:

$$cred_{op}^{p,\max} = I_{op}^{p} \cdot \frac{\sum_{k=0}^{p-1} cred_{op}^{k,\min} - L_{op}^{>p,\max}}{\sum_{k=0}^{p-1} I_{op}^{k} - C} \tag{3.2}$$

And $cred_{op}^{p,\min}$ is the credit lower bound:

$$cred_{op}^{p,\min} = (I_{op}^{p} - C) \cdot \frac{L^p}{C} \tag{3.3}$$

Note that the service curve for a CBS instance placed at a given priority p depends on the IdleSlope assigned to that priority, as well as the IdleSlopes of higher priority queues. This interdependence is of importance when considering CBS deployment, as the configuration of one CBS instance can impact the configuration of others in the output port.

3.2 Motivation for Partial CBS Deployment

In this section, we discuss the motivation for Partial CBS Deployment. We first highlight the cost-effectiveness of this approach, followed by an analysis of the blocking effect introduced by CBS in a Full Deployment scenario. Finally, we present a motivational example to illustrate

the challenges and considerations for implementing a Partial CBS Deployment strategy.

3.2.1 Partial CBS Deployment is cost-effective and Lowers Design and Operation Complexity

As discussed in Chapter 2, the main motivation for Partial CBS Deployment stems from the interoperability challenges between legacy Ethernet switches and TSN-capable ones. The need for this interoperability arises from the high costs associated with upgrading existing network infrastructure to support TSN standards. Replacing all legacy switches with TSN-capable ones can be prohibitively expensive and logistically challenging.

The idea behind Partial CBS Deployment is to identify the most critical points in the network where CBS can provide the most benefit. By strategically deploying CBS only where it is most needed, we can optimize resource utilization and ensure that the network meets its performance requirements without unnecessary overhead. This targeted approach allows for a more efficient use of network resources, ensuring that high-priority traffic receives the necessary QoS while minimizing the impact on overall network delays. Therefore, by developing a methodology to identify these critical points, we inherently understand where CBS is most needed.

One motivating, more quantitative factor for partial deployment is to mitigate the blocking effect that CBS can introduce for shaped traffic. To perform effective partial CBS deployment, all flows must be schedulable; therefore, understanding how CBS impacts flow schedulability is crucial. In this section, we analyze the blocking effect introduced by CBS and how partial deployment can help mitigate it.

3.2.2 Delay improvement of high-priority shaped traffic through Partial CBS Deployment

The blocking effect on high-priority shaped traffic was first discussed by Diemer et al. [96]. They observed that in a mixed-priority scenario, where high-priority shaped and medium-priority unshaped traffic coexist, the shaped traffic can experience delays due to the CBS credit mechanism.

Specifically, when a high-priority frame is ready to transmit but its credit is negative (due to previous transmissions), it must wait until the credit becomes non-negative again. This waiting period allows lower-priority frames to be transmitted, leading to increased latency for the high-priority traffic.

This is illustrated in Figure 3.2. The figure assumes an output port similar to Figure 3.1, with three priorities and CBS shaping on $p = 0$. At $t = 0$, frames for priority $p = 0$ (high) and $p = 1$ (medium) arrive, while a Best Effort frame ($p = 2$) is transmitting. The credit for $p = 0$, which starts at 0, increases between points (a) and (b) as conflicting frames transmit. Credit then becomes positive, and $p = 0$ frames are sent, as they are the highest priority

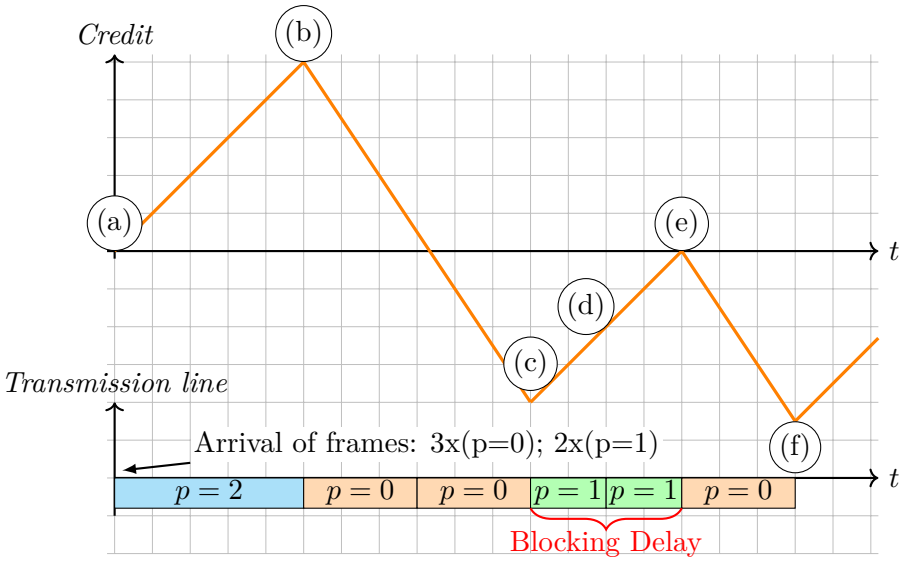


Figure 3.2: Illustration of the CBS blocking effect

frames in queue. However, the credit becomes negative before all $p = 0$ frames are sent. This negative credit forces the remaining $p = 0$ frames to wait, allowing $p = 1$ frames to transmit in the meantime, thus creating a blocking delay.

An interesting observation is that this blocking delay is directly related to the IdleSlope; an incorrectly dimensioned IdleSlope can exacerbate the blocking effect, as credit recovery would take longer.

Extending the blocking delay to the Full CBS Deployment paradigm, the blocking delay can accumulate over multiple CBS instances along a flow's path. Each CBS instance can introduce its own blocking delay, leading to a cumulative effect that can significantly impact the end-to-end delay of high-priority shaped traffic.

Therefore, partial CBS deployment not only helps with legacy interoperability and cost reduction but can also be a key strategy to reduce the blocking effect on high-priority shaped traffic.

3.2.3 Challenges and Considerations for a Partial CBS Deployment Strategy

We now provide a motivational example to illustrate the challenges and considerations for partial CBS deployment.

Successfully implementing Partial CBS deployment requires two main considerations:

1. **Placement:** Where to place CBS instances so that they have the greatest positive impact.

2. **Configuration:** Which IdleSlope to choose for each instance.

We use the following illustrative example to understand these factors.

Assume the network shown in Figure 3.3, traversed by the flows in Table 3.2. The network bandwidth is 1Gbps for all ports.

- **Flow** f_0: High-priority $p = 0$ shaped traffic.
- **Flow** f_1: Medium-priority $p = 1$ unshaped traffic.
- **Flow** f_2: Low-priority $p = 2$ unshaped traffic.

The paths were chosen to cause maximum interference from f_0 to f_1, and f_2 emulates background traffic.

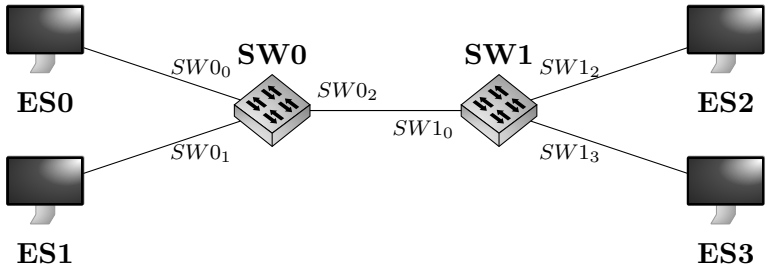


Figure 3.3: Topology illustrating the blocking effect of CBS on high-priority shaped traffic.

Table 3.2: Flow characteristics for the example network in Figure 3.3.

Flow	Source	Destination	Priority	Period (ms)	Deadline (ms)	Max Frame Size (B)	Burst (# of Frames)
f_0	ES0	ES2	0	1	1	300	15
f_1	ES1	ES2	1	1	1	300	1
f_2	ES3	ES2	2	100	100	1500	200

3.2.3.1 CBS Placement

To understand placement, we first analyze the end-to-end delay bound of the medium-priority flow (f_1) under four scenarios, keeping the IdleSlope fixed at $I^p_{op} = 63$Mbps (1.75 times the bandwidth requirement of f_0).

1. **Full CBS Deployment:** CBS at $SW0_2$ and $SW1_2$.
2. **Partial CBS:** Single CBS instance at $SW0_2$ (early in the path).

Table 3.3: End-to-end delay bounds for flow f_1 under different CBS deployment scenarios.

CBS Deployment Scenario	CBS Instance(s) Location(s)	End-to-End Delay Bound f_1 (ms)
Full CBS Deployment	$SW0_2, SW1_2$	0.870ms
Partial CBS Deployment	$SW0_2$	0.971ms
Partial CBS Deployment	$SW1_2$	1.296ms
No CBS (NP-SP only)	N/A	1.385ms

3. **Partial CBS:** Single CBS instance at $SW1_2$ (later in the path).

4. **No CBS:** Legacy Non-Preemptive Strict Priority (NP-SP) only.

Table 3.3 summarizes the delay bounds for flow f_1.

We observe that: No CBS fails to schedule f_1 (delay $>$ 1ms deadline) due to the high burstiness of f_0. Full CBS yields the lowest delay for f_1, as f_0's burstiness is addressed at both switches. Finally, Partial CBS demonstrates the importance of placement as placing CBS at $SW0_2$ successfully schedules f_1, whereas CBS at $SW1_2$ fails to schedule f_1. This hints to a placement strategy where CBS should be placed early in the path of high-priority shaped traffic to reduce its burstiness before it can interfere with medium-priority unshaped traffic. This also shows the ability of the partial CBS deployment approach to solve schedulability issues caused by the burstiness of traffic traversing the network.

3.2.3.2 IdleSlope Selection

IdleSlope selection is also of importance. To illustrate this, we analyze the impact of different IdleSlope values on the end-to-end delay bound of the high-priority flow (f_0), as shown in Figure 3.4.

From the figure, we observe that the end-to-end delay bound of f_0 decreases as the IdleSlope increases. This is expected, as a higher IdleSlope reduces the blocking effect on f_0. Conversely, a lower IdleSlope, which increases f_0's delay means increased access to the transmission line for f_1. Lastly, the figure shows that partial CBS scenarios are able to achieve schedulability for f_0 at a lower IdleSlope value compared to the full deployment.

This highlights the central challenge of selecting an appropriate IdleSlope, which presents a critical trade-off. A low IdleSlope value enhances medium-priority traffic schedulability but exacerbates the blocking effect on high-priority traffic. Conversely, a high IdleSlope mitigates this blocking but can, in turn, starve medium-priority traffic. This is precisely where partial CBS deployment demonstrates its potential. By reducing the cumulative blocking effect that arises from multiple shapers in series, a partial deployment strategy enables the use of lower IdleSlope values while still meeting high-priority deadlines. This capability directly improves the schedulability of medium-priority traffic, achieving the overall goal: identifying the lowest

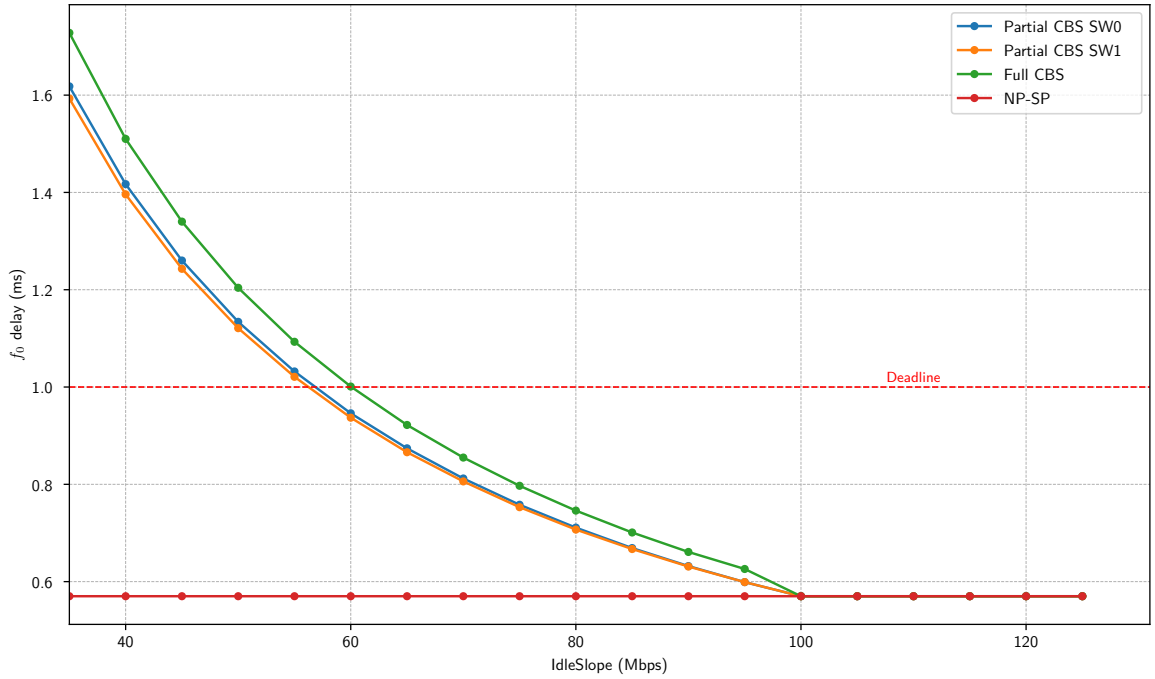


Figure 3.4: Impact of IdleSlope on flow f_0 delay

possible IdleSlope that ensures high-priority traffic schedulability in a network designed to schedule all traffic.

We formalize these challenges in the next section, defining the Partial CBS Deployment Problem.

3.3 The Partial CBS Deployment as an Optimization Problem

From Section 3.2, we understand that partial CBS deployment can provide significant benefits in terms of cost reduction, legacy interoperability, and performance improvement. In this section, we provide a formalization of the Partial CBS Deployment Problem as an optimization problem.

3.3.1 Problem Formulation

Objective Function: Minimizing the number of TSN devices required The problem we aim to solve is a placement problem. Specifically, we aim to determine strategic locations in the network to deploy CBS instances to ensure the best legacy network interoperability while ensuring the network remains schedulable. Thus, we aim for an approach enabling CBS clustering. The idea behind clustering CBS instances is to reduce the number

of devices that need to support TSN features, thus lowering deployment costs and complexity. By grouping CBS instances[3] on fewer devices, we can achieve the desired traffic shaping and scheduling benefits while minimizing the overall impact on the network infrastructure. This way, the partial deployment of CBS philosophy is fully respected, as we only deploy CBS where it is most needed, while ensuring that the network remains schedulable and meets its performance requirements. To this end, we introduce variable TSN_{dev}, which represents if a device implements CBS functionality. This is a binary variable that takes 1 if at least a CBS is implemented in *dev* and 0 if none are implemented. This variable is defined as follows:

$$TSN_{dev} = \begin{cases} 1 & \text{if } \sum_{\forall op,p|op \in OP(dev)} CBS_{op}^{p} \geq 1 \\ 0 & \text{if } \sum_{\forall op,p|op \in OP(dev)} CBS_{op}^{p} < 1 \end{cases} \tag{3.4}$$

The optimization problem is then defined as the minimization of the sum of all TSN_{dev} variables, ensuring that the network remains schedulable. This leads to the following objective function:

$$\text{minimize} \sum_{\forall dev \in DEV} TSN_{dev} \tag{3.5}$$

Priority Constraint For the constraints, we need to ensure we follow both the standard and its recommendations, as well as ensuring real-time network guarantees such as schedulability and stability. Regarding the standard and its recommendations, in [97], we find note 5, reproduced below:

> **Note 5 from IEEE 802.1Q-2022 [97].** In order for the CBS algorithm to operate as intended, it is necessary for all traffic classes that support the algorithm to be numerically higher than any traffic classes that support the strict priority algorithm.

The interpretation of this note is that any given priority-queue where CBS is implemented must have a higher priority than any traffic classes not implementing it.

In addition to this, as stated in Equation 3.2, the credit upper bound of a CBS instance depends on the IdleSlopes of higher priority queues. Therefore, to ensure the correct operation of CBS, we need to ensure that if CBS is implemented for a given priority p, then it must also be implemented for all higher priorities $p' < p$. This is expressed through the following constraint:

$$CBS_{op}^{p} \geq CBS_{op}^{p+1}, \forall op \in OP, \forall p \in P. \tag{3.6}$$

IdleSlope Upper Bound Moreover, in IEEE 802.1Qav [32], it is stated that traffic classes implementing CBS in practice should use at most an IdleSlope such that the aggregation of the used IdleSlopes is at most 75% of the output port's bandwidth. The rationale behind this recommendation is to prevent the network starvation issues CBS is meant to avoid. The motivation is that CBS should work in synergy with the Stream Reservation Protocol

[3] A CBS instance refers to one CBS implemented for a given priority-queue of a given output port.

(SRP), which is a mechanism defined in IEEE 802.1Qat [85] and allows for decentralized flow allocation. SRP defines two classes SR class A and SR class B, both with stringent delay requirements. Thus, to ensure these are shaped by CBS, but still having a sufficiently high IdleSlope, a practical limit between these two classes has been set. This is expressed through a note in Annex L of the standard, reproduced below:

> **Note from IEEE 802.1Qav-2009 [32].** In general, if SR Class A is permitted to use $X\%$ of the bandwidth, then only $(100-X)\%$ remains available for use by lower priority traffic classes. In practice, as 75% is the default value for the highest percentage of portTransmitRate that can be used by all SR classes, there is always at least 25% of portTransmitRate available for use by the non-SR traffic classes.

Abstracting from SRP, we can interpret this note as a practical recommendation to ensure that the sum of all IdleSlopes of CBS implemented in an output port is lower than 75% of the output's bandwidth. This ensures that there is always at least 25% of the bandwidth available for non-CBS traffic, thus preventing starvation issues. This can be formalized through the following constraint:

$$\sum_{p \in P} I^p_{op} \cdot CBS^p_{op} \leq 0.75 \cdot C, \quad \forall op \in OP. \tag{3.7}$$

Network Stability Condition In addition, we aim to ensure CBS placement does not affect the stability or schedulability of the network. Stability can be kept by ensuring the arrival rate of traffic to a priority queue is lower than the rate at which it is served. This can be observed through a simple example. Assume a priority queue p with an arrival curve defined through the leaky-bucket form $\gamma_{r,b}$. Assume the queue is served by a Rate-Latency Service Curve $\beta_{R,T}(t)$[4]. If the rate from the Arrival Curve r is greater than the service rate from the Service Curve R (i.e. $r > R$), then it is impossible to apply Theorem 1.4.2 of [51] to compute a delay bound, as the horizontal deviation between the arrival curve and the service curve is infinite. Thus, to ensure stability, we need to ensure that for any given priority queue p with CBS implemented, the assigned IdleSlope I^p_{op} respects the stability condition of the priority level, i.e., $r^p \leq I^p_{op}$.

In the case CBS were to be implemented, this condition would be expressed as:

$$I^p_{op} \geq r^p_{op} \cdot CBS^p_{op}, \quad \forall op \in OP, \forall p \in P \tag{3.8}$$

Proof. The proof for Equation 3.8 follows from the Service Curve a CBS and NP-SP scheduler offer to a priority queue p. Assuming the aggregate Arrival Curve to priority queue p is given by $\alpha^p_{op}(t) = r^p_{op}t + b^p_{op}$, where r^p_{op} is the sum of the rates of all flows assigned to priority p. If CBS is implemented for priority p. Then acknowledging that the service Rate R is given by the IdleSlope I^p_{op}, from Equation 3.1, the stability condition is expressed by Equation 3.8. □

[4]Both presented in Appendix A

Schedulability Constraint Regarding the schedulability condition, we need to ensure that the end-to-end delay of all flows is lower than their respective deadlines. This can be expressed as:

$$d^f \leq D^f, \forall f. \tag{3.9}$$

Where d^f is the end-to-end delay bound of flow f, computed using the TFA++ methodology described in Chapter A.6.4.

Here d^f is defined as:

$$d^f = \max_{\forall f_i} \left(d^f_{src} + d_{prop} + \sum_{\forall k \in \phi^{f_i} \setminus src} d^f_k \right) \tag{3.10}$$

IdleSlope Computation Strategy One final consideration is the selection of IdleSlope values. We therefore choose, for each CBS instance, the smallest IdleSlope that guarantees both the stability and the schedulability of the shaped classes (as discussed in Section 3.2). Minimizing IdleSlopes while still ensuring shaped traffic schedulability is the best trade-off at improving fairness while preserving the required network guarantees; consequently, the optimization seeks to minimize IdleSlopes subject to the stability and schedulability constraints for all shaped classes.

With these considerations in mind, we can now formally define the optimization problem we aim to solve.

Objective

$$\text{minimize} \quad \sum_{dev \in DEV} TSN_{dev} \tag{3.11}$$

Subject to

$$\sum_{p \in P} I^p_{op} \cdot CBS^p_{op} \leq 0.75 \cdot C, \qquad \forall op \in OP \tag{3.12}$$

$$CBS^p_{op} \geq CBS^{p+1}_{op}, \qquad \forall op \in OP, \forall p \in P \tag{3.13}$$

$$I^p_{op} \geq r^p \cdot CBS^p_{op}, \qquad \forall op \in OP, \forall p \in P \tag{3.14}$$

$$d^f \leq D^f, \qquad \forall f \in \mathcal{F} \tag{3.15}$$

$$CBS^p_{op} \in \{0, 1\}, \qquad \forall op \in OP, \forall p \in P \tag{3.16}$$

Where the optimization variables are CBS^p_{op} and I^p_{op} for all output ports op and priorities p.

As presented, this problem corresponds to a Non-Linear Programming (NLP) optimization

problem, that is commonly known to be hard to solve [98]. This comes from the non-linearity introduced by the objective function, specified in Equation 3.5. As it depends on the binary variable TSN_{dev}, which itself depends on the binary variables CBS^p_{op} through Equation 3.4, it introduces a non-linearity in the objective function.

Another complexity is due to Equation 3.15 relying on an oracle's verification. This equation is also non-linear due to the IdleSlope being in the denominator of the per-node delay computation (See Equations 3.1 and 3.10). Moreover, the oracle, which implements the FP-TFA++ algorithm, is recursive in nature, which is time-consuming and exacerbates the non-linearity of Constraint 3.15.

To address these complexities, we conduct further relaxation and linearization techniques to obtain a linear sub-problem that is easier to solve. This sub-problem is then used as the basis for a heuristic-based framework presented in Section 3.4.1.

3.3.2 Problem Linearization and Relaxation

We linearize Constraint 3.16 by replacing it with the following linear constraints:

$$\begin{aligned} &TSN_{dev} \geq \mathrm{CBS}^p_{op}, && \forall dev \in DEV, \forall (op,p) \,|\, op \in OP(dev) \\ &CBS^p_{op} \in \{0,1\} && \forall op, p \\ &TSN_{dev} \in \{0,1\} && \forall dev \in DEV \end{aligned} \tag{3.17}$$

Second, as discussed, the non-linearity of Equation 3.15 is mainly due to the recursive way of computing it, as it requires the computation of burst propagation along the flows paths (See SectionA.6.4). Originally, the delay per port for an arrival curve $\alpha^{pred}(t)$, composed of all the output arrival curves from preceding nodes and service curve $\beta(t)$ is defined as:

$$d^p_{op} = h(\alpha^{pred}(t), \beta(t)) \tag{3.18}$$

Where h corresponds to computing the maximum horizontal deviation between the two curves. In Equation 3.18, the burst propagation is obtained by computing $\alpha^{pred}(t)$. Computing this arrival curve requires obtaining the output arrival curve of all preceding nodes. In a situation for which a preceding node has a leaky bucket arrival curve and a Rate Latency service curve, the output arrival curve is defined as: $rt+rT+b$. Here, rT constitutes the burstiness increase. This has to be done recursively, per flow to obtain the complete arrival curve at the node of interest. (See A.6.4).

Hence, to remove this recursiveness, the constraint is relaxed by ignoring the burst propagation and considering the initial burst size of flows generated by the source. This relaxation allows for the computation of the minimum upper delay bound for each traffic class with priority p within each output port op, $\underline{d}^p_{op}$ as follows:

$$\underline{d}_{op}^{p} = \frac{\sum_{\forall f \in \mathcal{F}(op,p)} b^{f}}{R_{op}^{p}} + T_{op}^{p} \tag{3.19}$$

Where, R_{op}^{p} and T_{op}^{p} are respectively the service rate and latency computed using Equation 3.1 or Theorem A.5, depending on whether CBS is implemented for the (op, p) of interest.

As a reminder we reproduce below the definitions of R_{op}^{p} and T_{op}^{p} in either case.

In the case of CBS implementation:

$$R_{op}^{p} = I_{op}^{p}$$

$$T_{op}^{p} = I_{op}^{p} \cdot \frac{\sum_{k=0}^{p-1} cred_{op}^{k,\min} - L_{op}^{>p,\max}}{\sum_{k=0}^{p-1} I_{op}^{k} - C}$$

Where $cred_{op}^{p,\min}$ is the credit lower bound:

$$cred_{op}^{p,\min} = (I_{op}^{p} - C) \cdot \frac{L^{p}}{C}$$

And in the case CBS is not implemented for the queue:

$$R_{op}^{p} = C - \sum_{p=0}^{p-1} r_{op}^{p}$$

$$T_{op}^{p} = \frac{\sum_{p'=0}^{p-1} b_{op}^{p'} + L_{op}^{>p,max}}{C - \sum_{p'=0}^{p-1} r_{op}^{p'}}$$

Afterward, instead of verifying the Equation 3.15 based on the end-to-end delay bound and deadline of a flow, we simplify the problem by verifying this constraint locally for each output port and priority level. This is done by defining a local deadline D_{op}^{p} for each op and p. The local deadline is derived from the end-to-end deadlines of all flows traversing the output port op with priority p, based on a proportion of the arrival rate for p at op. This leads to the following definition of the local deadline:

$$D_{op}^{p} = \min_{\forall f \in \mathcal{F}(op,p)} \left[\left(D^{f} - d_{src}^{p} - d_{prop} \right) \cdot \frac{r_{op}^{p}}{\sum_{op' \in \phi^{f_i} \setminus src} r_{op'}^{p}} \right] \tag{3.20}$$

Proof. By substituting Equation 3.10 into Equation 3.15, we obtain for any $f \in \mathcal{F}(op, p)$

$$\sum_{\forall op \in \phi^{f} \setminus src} d_{op}^{p} \leq D^{f} - d_{src}^{p} - d_{prop}$$

There exists a local deadline D^p_{op} such that:

$$\sum_{\forall op \in \phi^f \setminus src} d^p_{op} \leq \sum_{\forall op \in \phi^f \setminus src} D^p_{op} \leq D^f - d^p_{src} - d_{prop} \tag{3.21}$$

Consequently, if we consider

$$\sum_{\forall op \in \phi^f \setminus src} D^p_{op} \leq \min_{f \in \mathcal{F}(op,p)} \left[D^f - d^p_{src} - d_{prop} \right]$$

Then, we can compute D^p_{op} as:

$$D^p_{op} = \min_{f \in \mathcal{F}(op,p)} \left[(D^f - d^p_{src} - d_{prop}) \cdot \frac{r^p_{op}}{\sum_{\forall op' \in \phi_f \setminus src} r^p_{op'}} \right]$$

Summing D^p_{op} over all output ports in the path of any flow $f \in \mathcal{F}(op, p)$ (excluding the source) verifies Equation 3.21. This completes the proof. □

The local deadline is determined using a heuristic that allocates more time to the network's busiest points. This strategy assigns the longest local deadline to the most utilized port, i.e., the bottleneck along a flow's path. The calculation considers two factors at a given output port *op*: the cumulative traffic rate from flows with a specific priority p and the most stringent deadline among all flows crossing that port. This approach improves the likelihood of meeting scheduling constraints precisely where traffic congestion is highest.

We have now the elements to define a local schedulability condition. From Equations 3.19 and 3.20, a local schedulability condition is derived for $\forall op \in OP, \forall p \in P$ in Equation 3.22.

$$\underline{d}^p_{op} \leq D^p_{op} \cdot \text{margin} \tag{3.22}$$

Here the variable $margin \in [0, 1]$ is introduced herein to compensate the fact of ignoring burst propagation.

Proof. Assume a lower bound for margin of:

$$\text{margin} \geq 1 - \frac{d^p_{op} - \underline{d}^p_{op}}{d^p_{op}}$$

Then, from Equation 3.15 and the upper bound above, we know the following expression to be true:

$$d^p_{op} \cdot \left(1 - \frac{d^p_{op} - \underline{d}^p_{op}}{d^p_{op}} \right) \leq D^p_{op} \cdot \text{margin}.$$

The expression above can be simplified into:

$$d^p_{op} - \left(d^p_{op} - \underline{d}^p_{op}\right) \leq D^p_{op} \cdot \text{margin}$$

$$\underline{d}^p_{op} \leq D^p_{op} \cdot \text{margin}$$

Thus completing the proof. □

Starting with a value of 1, in case burst propagation is significant, the variable *margin* can be adjusted to a lower value, artificially increasing the computed local delay, to ensure the local schedulability condition remains valid.

The different linearization and relaxation considerations above lead to the following linear optimization sub-problem:

Objective

$$\text{minimize} \quad \sum_{dev \in DEV} TSN_{dev} \tag{3.23}$$

Subject to

$$\sum_{p \in P} I^p_{op} \cdot CBS^p_{op} \leq 0.75 \cdot C, \qquad \forall op \in OP \tag{3.24}$$

$$I^p_{op} \geq r^p \cdot CBS^p_{op}, \qquad \forall op \in OP, \forall p \in P \tag{3.25}$$

$$TSN_{dev} \geq CBS^p_{op}, \qquad \forall dev \in DEV, \forall op \in OP(dev), \forall p \in P \tag{3.26}$$

$$\underline{d^p_{op}} \leq D^p_{op} \cdot \text{margin}, \qquad \forall op \in OP, \forall p \in P \tag{3.27}$$

$$CBS^p_{op} \in \{0, 1\}, \qquad \forall op \in OP, \forall p \in P \tag{3.28}$$

$$TSN_{dev} \in \{0, 1\}, \qquad \forall dev \in DEV \tag{3.29}$$

$$CBS^p_{op} \geq CBS^{p+1}_{op}, \qquad \forall op \in OP, \forall p \in P \tag{3.30}$$

Where, Constraints 3.26, 3.28, and 3.29 correspond to the linearization of Constraint 3.16. Constraint 3.27 corresponds to the local schedulability condition derived from Constraint 3.15.

It is worth noting that in time-sensitive networks with more than 10 devices and more than 1000 flows, the number of constraints to produce a feasible solution can exceed 10000. Known ILP solvers [99] are computationally inefficient for such large-scale problems. Therefore, we choose to address this problem with a custom heuristic approach that is introduced and detailed in Section 3.4.

3.4 Framework for Partial CBS Deployment

In this section, we present our framework for optimizing the partial deployment of CBS in Time-Sensitive Networks. As discussed, using traditional optimization techniques might be inefficient. Due to the modifications done to the optimization problem, we must first solve it through a heuristic and then verify the solution with an NC tool. The mixture of these two processes composes the framework. The heuristic is designed to strategically place and configure CBS instances to ensure network schedulability while minimizing the number of devices requiring TSN capabilities. Then a NC-based Schedulability Analyzer validates the proposed configuration. This process and the details of the heuristic are presented in the following sections.

3.4.1 Framework Overview for Optimizing Partial CBS Deployment

Our framework presents a systematic method for strategically placing and configuring CBS within a Time-Sensitive Network with legacy Ethernet compatibility requirements. The primary objective is to guarantee that all traffic flows meet their critical deadlines while minimizing the number of devices that need costly TSN capabilities.

As illustrated in Figure 3.5, this optimization is achieved through the interplay of two core components: the **Burstiness-aware Partial CBS Deployment Algorithm** that proposes a solution and a **Schedulability Analyzer** that validates it.

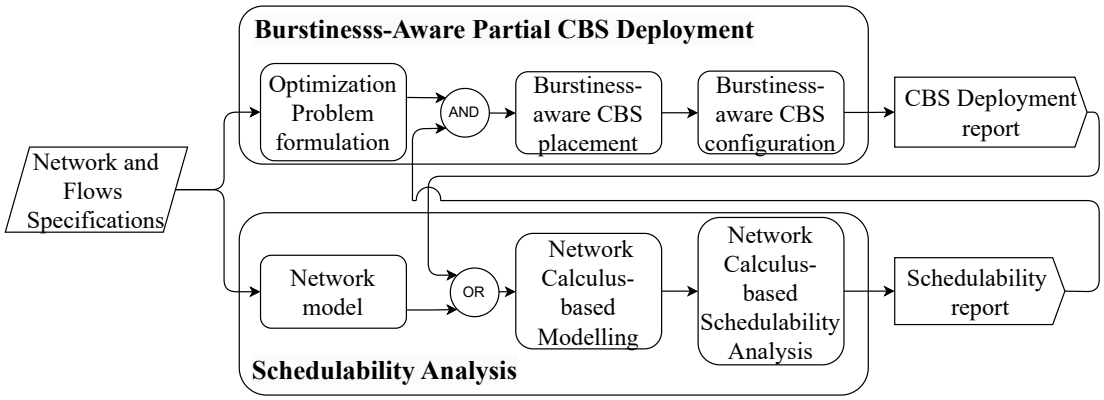


Figure 3.5: Partial CBS Deployment Framework

The Burstiness-aware Partial CBS Deployment Algorithm: This component is the core of the optimization process, responsible for determining the most effective placement and configuration of CBS. Figure 3.5 presents a flowchart detailing the global operation of the algorithm. It works with the input of the *Schedulability Analyzer*. If it receives a configuration that meets schedulability constraint of flows assigned to the highest unshaped priority but with unscheduled lower priority flows, then it generates placement and configuration for CBS

instances clustered to devices of choice, which is then fed to the *Schedulability Analyzer*. In contrast, if the configuration received fails to meet the schedulability of highest unshaped priority traffic, then it adjusts the *margin* parameter for the already placed CBS instances to be more conservative, and generates a new candidate solution. This process continues until a satisfactory configuration is found. We call this algorithm the *Burstiness-aware Partial CBS Deployment Algorithm*, as it strategically places CBS instances at the places where burstiness is the highest, as we will detail in Section 3.4.2.

The Schedulability Analyzer: The core of the validation engine is the schedulability analyzer, which is implemented as an instance of the WOPANets tool [100]. As the solution given is based on a relaxation of the original problem, it is crucial to verify that the proposed CBS placement and configuration indeed ensures that all flows meet their deadlines. To do so, the analyzer employs the FP-TFA++ algorithm (presented in Section A.6.4), allowing a fast and scalable computation of end-to-end delay bounds and schedulability. If all flows are found to be schedulable under the proposed configuration, the analyzer confirms the solution's validity. However, if any flow fails to meet its deadline, it signals the Burstiness-aware Partial CBS Deployment Algorithm to adjust its parameters and generate a new configuration.

Now that we have presented the overall framework, we proceed to detail the heuristic algorithm used for CBS placement and configuration.

3.4.2 Burstiness-aware CBS Placement

Our heuristic for placing a CBS targets the switch containing the earliest point of contention between high-priority shaped traffic and medium-priority non-shaped traffic. This approach is motivated by the serialization effect [66], where the negative impact of a flow's burstiness is most pronounced at its first hop. Placing a CBS at this initial contention point allows for immediate mitigation of such burstiness.

The selection process is driven by prioritizing unscheduled flows whose delay bound is further from their deadline. This process is presented in Figure 3.6. In the flowchart in the Figure, the algorithm first compiles a list of all unscheduled and unshaped flows that do not belong to the highest unshaped priority. This list is sorted in ascending order based on each flow's difference between its computed end-to-end delay and its deadline ($d^f - D^f$). Flows with the highest difference are considered the most critical to schedule and are positioned at the top. In this list, the top flow is referred to as the most constrained flow.

The algorithm then proceeds iteratively. It selects the flow of interest (foi) from the top of the list and places CBS instances at the first device where flow foi meets higher priority unshaped traffic. A CBS is placed at every output port traversed by unscheduled flows within the selected device. Subsequently, all other flows in the list that are affected, directly or indirectly by the placed CBS are removed. Where *directly affected* refers to unscheduled flows that traverse the device and ports where a CBS was placed inside the device, while *indirectly affected* refers to unscheduled flows that do not traverse the selected device but

share parts of their path with flows in from the *indirectly affected* set or with the newly shaped flows whose paths intersect with at least one flow in *directly affected* flows set at any point downstream of the CBS placement.

To formalize the *directly affected* and *indirectly affected* sets, we consider the set of unscheduled non shaped flows, `unschedNonShaped`. The *directly affected* (DA) flows are then obtained with:

$$DA = \{f \mid f \in (\texttt{unschedNonShaped} \cap \mathcal{F}(\texttt{dev}))\} \tag{3.31}$$

And the *indirectly affected* (IA) flows with:

$$\begin{aligned} IA = \{&f \in \texttt{unschedNonShaped} \mid \\ &\forall f \notin \mathcal{F}(\texttt{dev}) \wedge (\exists f' \in DA \cup \texttt{shapedFlows} \mid \phi^f \cap \phi^{f'} \neq \emptyset)\} \end{aligned} \tag{3.32}$$

This process is repeated with the new top-most flow until the list is empty, thereby systematically addressing potential schedulability issues throughout the network. The entire process is detailed in Figure 3.6. The main algorithm is detailed in Algorithm 1 in Section 3.4.4.

Note that Equation 3.1 shows that the service provided by a CBS instance depends on the IdleSlope of higher priority CBS instances. Thus, placing CBS instances at multiple priorities within the same output port can lead to sub-optimal IdleSlope configurations. By limiting the placement to a single CBS per output port per iteration, we avoid this complexity and ensure a more straightforward and effective configuration process. Through this process, we also ensure Constraint 3.30 is respected, as CBS instances are placed sequentially from the highest to the lowest priority.

3.4.3 Burstiness-Aware CBS Configuration

Once we know the devices where to place CBS instances, they must be configured. In Section 3.3, the challenge of selecting appropriate IdleSlope values was discussed. In general terms, the selected IdleSlope for a given CBS instance should be the lowest possible value that still ensures the schedulability of the shaped class. The rationale behind this choice is to minimize the blocking effect on high-priority shaped traffic while benefiting the schedulability of lower priority unshaped traffic. Therefore, to compute the IdleSlope at a given output port op and priority p where we wish to place a CBS, we start from the local schedulability condition defined in Equation 3.27. Since the burst propogation from FP-TFA++ was removed, we are able to provide a closed-form expression for the minimum IdleSlope.

From Theorem A.2, we know that the delay bound is computed as the maximum horizontal deviation between an arrival curve and a service curve. In our case, the arrival curve is defined through a leaky-bucket form, while the service curve is defined through a Rate-Latency form. Thus, the local delay bound $\underline{d}^p_{op}$ results in Equation 4.2. From there, assuming a CBS is placed at (op, p), i.e., $\text{CBS}^p_{op} = 1$, we can compute the service curve using Equation 3.1. In

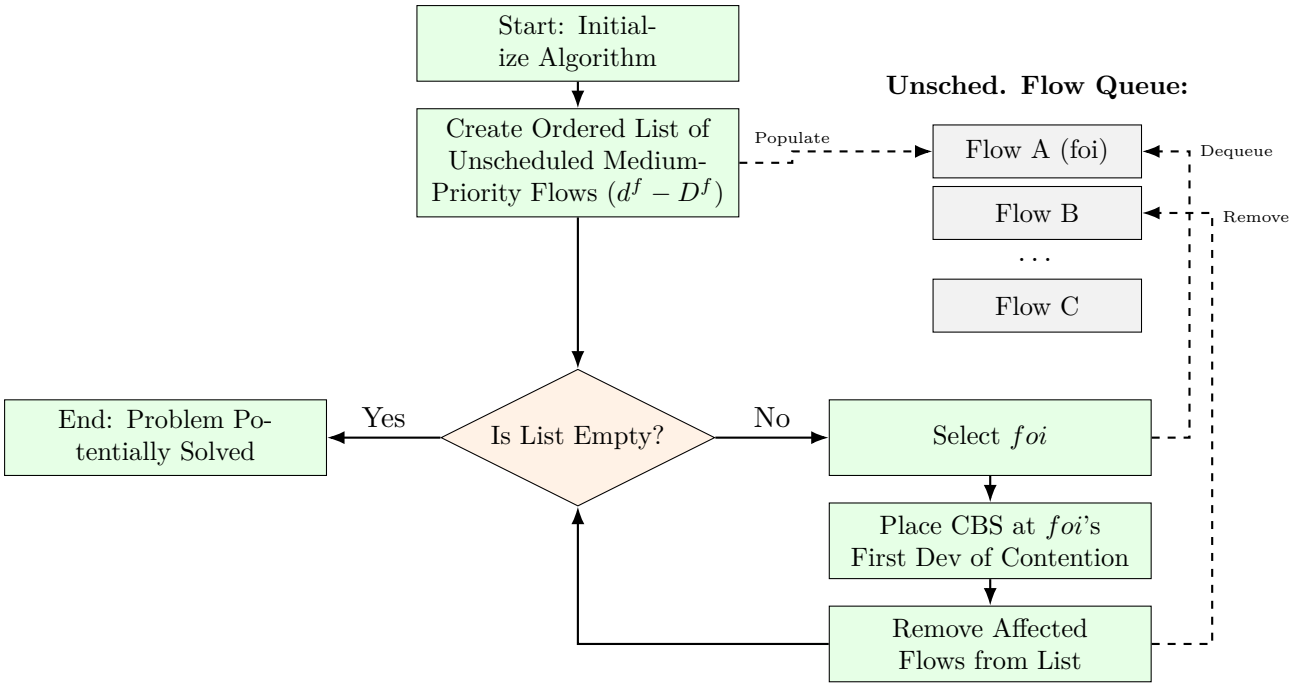


Figure 3.6: Flowchart detailing device selection for burstiness-aware partial CBS placement.

this case, the local delay bound from Equation 4.2 becomes:

$$\frac{\sum_{f\in\mathcal{F}(op,p)} b^f}{I^p_{op}} + \frac{\sum_{k=0}^{p-1} cred^{k,\min}_{op} - L^{>p,\max}_{op}}{\sum_{k=0}^{p-1} I^k_{op} - C} \leq D^p_{op} \cdot \text{margin} \tag{3.33}$$

Where D^p_{op} is computed with Equation 3.20, and $L^{>p,\max}_{op}$ is the maximum frame size of the lower priority interfering within port op.

We can then rearrange Equation 3.33 to isolate I^p_{op} on the left-hand side, thus obtaining a closed-form expression for the minimum IdleSlope at (op, p) that ensures the local schedulability condition is met:

$$I^p_{op} \geq \frac{\sum_{f\in\mathcal{F}(op,p)} b^f}{D^p_{op} \cdot margin - \frac{\sum_{k=0}^{p-1} cred^{k,\min}_{op} - L^{>p,\max}_{op}}{\sum_{k=0}^{p-1} I^k_{op} - C}} \tag{3.34}$$

Now that we have a closed-form bound for the minimum IdleSlope, we complete the minimum Idleslope computation by ensuring that the stability condition defined in Equation 3.8 is also met. Thus, the final expression for the minimum IdleSlope at (op, p) becomes:

$$I^{p,min}_{op} = \max\left(\frac{\sum_{f\in\mathcal{F}(op,p)} b^f}{D^p_{op} \cdot margin - \frac{\sum_{k=0}^{p-1} cred^{k,\min}_{op} - L^{>p,\max}_{op}}{\sum_{k=0}^{p-1} I^k_{op} - C}}, r^p_{op}\right) \tag{3.35}$$

Where I_{op}^{k} for $k < p$ are the IdleSlopes already configured for higher priority CBS instances at output port *op*.

Finally, regarding the *margin* parameter, it is initialized to 1 at the beginning of the algorithm. At this value, IdleSlope computation might render shaped flows unschedulable. Since *margin* is meant to compensate the absence of burstiness increase, its value is decreased by a constant rate ρ, until IdleSlope computation is able to schedule shaped flows. This is done until a minimum theoretical margin is reached, over this value, we consider using CBS on the selected output port is not possible. Note that margin is a global parameter, meaning the same value is used for every IdleSlope computation.

This minimum theoretical margin depends on all output ports and priorities where we place CBS, and it is computed as:

$$margin_{min} = \max_{\forall op \in OP, \forall p \in P} \left(\frac{\underline{d}_{op}^{p}}{D_{op}^{p}} \right) \tag{3.36}$$

Proof. The proof for Equation 3.36 follows from the local schedulability condition defined in Equation 3.27. For any output port *op* and priority p, to ensure the local schedulability condition is met, the following must hold:

$$\underline{d}_{op}^{p} \leq D_{op}^{p} \cdot margin$$

Rearranging this inequality gives:

$$margin \geq \frac{\underline{d}_{op}^{p}}{D_{op}^{p}}$$

To ensure that the local schedulability condition is satisfied for all output ports and priorities where CBS is placed, we take the maximum of the right-hand side over all *op* and p:

$$margin_{min} = \max_{\forall op \in OP, \forall p \in P} \left(\frac{\underline{d}_{op}^{p}}{D_{op}^{p}} \right)$$

This ensures that the chosen margin is sufficient to satisfy the local schedulability condition across the entire network. □

With these elements, we are able to provide a complete methodology for CBS placement and configuration. The full algorithm is detailed in the next section.

3.4.4 Heuristic Algorithm

The Burstiness-aware Partial CBS Deployment Algorithm, detailed in Algorithm 1, strategically places and configures CBS in a network, as detailed in Sections 3.4.2 and 3.4.3. It takes four arguments: a margin variable m; its constant rate of decrease ρ; the sets of unscheduled shaped (*unschedShaped*) flows; unscheduled non-shaped (*unschedNonShaped*) flows; the

already explored devices from previous iterations (*devsToExclude*); and *foi_prev* which corresponds to the previously selected flow of interest by the previous iteration of the algorithm. For the first iteration, this value is null.

The algorithm returns a candidate network configuration with CBS placements and IdleSlope values or indicates failure if no feasible solution exists.

The algorithm operates in two main phases.

Phase 1: Reconfiguration

This phase is triggered if the previous CBS configuration resulted in any unschedulable shaped flows. Its purpose is to correct previously underestimated `IdleSlope` values (Lines 2–10).

The process begins by reducing the margin parameter, m, by a fixed rate ρ (Line 4). This new margin is then compared against the minimum theoretical margin as given in Equation 3.36 (Lines 5–7).

- If the adjusted margin falls below this theoretical limit, a feasible solution cannot be found, and the algorithm terminates (Line 6).
- Otherwise, the `recomputeIdleSlopes` function updates all `IdleSlope` values using the new, smaller margin (Line 8). The resulting configuration is then returned to be re-evaluated for schedulability (Line 9).

Phase 2: CBS Placement

This phase executes if the input network configuration has all shaped flows schedulable, but at least one non-shaped flow is not (Lines 11–33). The goal here is to deploy new CBS instances to handle these remaining unschedulable flows.

The process starts by initializing a list of devices to exclude from consideration, `devsToExclude`, and a list to track newly shaped flows, `shapedFlows` (Lines 11–12).

A `while` loop continues as long as the set of unschedulable non-shaped flows, `unschedNonShaped`, is not empty (Line 13). Inside the loop, the algorithm performs the following steps:

1. Select the Target Flow and Device, based in the description of Section 3.4.2

First, it selects the flow of interest (`foi`), defined as the flow with highest delay difference to its deadline, highest-priority, unschedulable non-shaped flow (Line 15). This can be formalized into:

$$foi = \arg \max_{f \in \texttt{unschedNonShaped}} \left(d^f - D^f \right) \tag{3.37}$$

Next, it identifies the device (`dev`) where `foi` first encounters unshaped, higher-priority traffic (Line 19). The output ports on this device are then sorted into a list (`ops`) based on the most constrained flow crossing each port (Line 20).

2. Attempt CBS Placement

The algorithm then iterates through the sorted ports (`ops`) to find a suitable location to place a new CBS (Lines 22–29). For each port op and the highest non-shaped priority p, it computes the minimum required `IdleSlope`, I_{op}^{p}, using Equation 3.34.

If this calculated I_{op}^{p} value satisfies Constraint 3.24 (Line 24), a CBS is deployed at that port and priority level (op, p), and the corresponding flow is added to the `shapedFlows` set (Line 27). If the constraint is violated, that specific port is simply ignored, and the algorithm moves to the next one.

3. Update the Set of Unscheduled Flows

After iterating through all ports of a device, the algorithm updates the `unschedNonShaped` set. It identifies and removes two groups of flows whose schedulability may have changed:

- **Directly Affected (DA):** Unscheduled flows that traverse the device `dev` and ports where a CBS was placed inside the device (Line 30). Computed with Equation 3.31.
- **Indirectly Affected (IA):** Unscheduled flows that do not traverse `dev` but share parts of their path with flows in DA or with the newly shaped flows (Line 31). Computed with Equation 3.32.

These flows are removed from `unschedNonShaped` because their schedulability must be re-assessed following the new CBS placement (Line 32).

Once the `unschedNonShaped` list is empty, the new network configuration is returned for a final schedulability verification (Line 34).

3.4.4.1 Example use of the algorithm

To illustrate how Algorithm 1 works, we use the network in Figure 3.3. Assume it is now traversed by flows from Table 3.4. This table also contains the flow's characteristics. Flows are assigned to one of three priority levels, for which $p = 2$ is considered best effort and the lowest priority.

Table 3.5 shows the initial delays first input to the Algorithm, computed with the Schedulability Analyzer from [100]. It also shows the resulting delays after running the algorithm

Algorithm 1 Partial CBS Algorithm

```
1: function PartialCBSDeployment(m,ρ,unschedShaped, unschedNonShaped, alreadyExploredDevs, foi_prev)
2:     if unschedShaped ≠ ∅ then
3:         // Perform Reconfiguration
4:         m ← m − ρ
5:         if m ≤ m_min then
6:             return null                                   ▷ No solution could be found
7:         end if
8:         recomputeIdleSlopes(m)
9:         return network configuration
10:    end if
11:    devsToExclude ← alreadyExploredDevs
12:    shapedFlows ← ∅
13:    while unschedNonShaped ≠ ∅ do
14:        // Place CBS
15:        foi ← MostConstrainedFlow(unschedNonShaped)
16:        if foi == foi_prev && (φ^f \ {src, dst}) ∈ devsToExclude then
17:            return null                                   ▷ No solution could be found
18:        end if
19:        dev ← getDev(foi, devsToExclude)
20:        ops ← getOrderedOps(dev)
21:        // Configure CBS
22:        for all op ∈ ops do
23:            p ← highestNonShapedPriority(op)
24:            I^p_op ← ComputeIdleSlope(op)
25:            if Σ_∀p I^p_op ≤ 0.75 · C then
26:                CBS^p_op ← 1
27:                shapedFlows.add({f | f ∈ F(op, p)})
28:            end if
29:        end for
30:        DA ← getDirectlyAffectedFlows()
31:        IA ← getIndirectlyAffectedFlows()
32:        unschedNonShaped.remove(DA ∪ IA)
33:    end while
34:    return network configuration
35: end function
```

Table 3.4: Flow Characteristics

Flow	P	b (kbit)	Period (ms)	r (Mbps)	Path
f_0	0	14.4	1	14.4	$ES_0 \rightarrow ES_2$
f_1	0	14.4	1	14.4	$ES_2 \rightarrow ES_1$
f_2	1	0.96	1	0.96	$ES_1 \rightarrow ES_2$
f_3	1	0.96	1	0.96	$ES_2 \rightarrow ES_1$
f_4	1	0.96	1	0.96	$ES_3 \rightarrow ES_2$
f_5	2	123.4	100	1.234	$ES_3 \rightarrow ES_2$

for one single iteration. These results show a schedulable configuration, where the initial configuration had issues scheduling (f_2, f_3 and f_4). The step-by-step execution of the algorithm is depicted below.

Table 3.5: Variables of interest to the CBS deployment algorithm from the verification tool

Flow	src Delay Bound (μs)	Delay bound (NP-SP) (μs)	Delay bound (Partial CBS) (μs)	Deadline (μs)
f_0	216	314	479	1000
f_1	285	213	378	1000
f_2	-	543	396	535
f_3	-	560	419	555
f_4	-	473	472	472
f_5	-	-	-	-

Step-by-Step Execution

The initial set of unscheduled non-shaped flows is $\{f_2, f_3, f_4\}$ and `margin` is set to 1. Below, we outline the line-by-line results of the algorithm's execution:

1. **Line 2**: List `unschedShaped` is initialized to an empty set ($\emptyset$)
2. **Line 13**: Then, since f_2 is the flow whose delay is furthest from its deadline (by $8\mu s$), it is selected as *foi*
3. **Line 15**: Along the path of f_2, $SW0$ is the first time it encounters higher priority traffic, thus this device is selected first.
4. **Line 20**: There are two ports with unscheduled traffic in $SW0$: $(SW0_2, SW0_1)$. These ports are ordered as such as f_2, which traverses $SW0_2$ has a higher delay slack than f_3.
5. **Lines 22–29 (First iteration of the `for` loop)**:
   - The first iteration of the for loop is with op$\leftarrow SW0_2$
   - Since we have not placed any CBS, p $\leftarrow 0$
   - Using Equation 3.35, we compute the respective IdleSlope $I^0_{SW0_2} = 44460912$bps, using $margin = 1$
6. **Lines 22–29 (Second iteration of the `for` loop)**:
   - The next iteration of the for loop is for op $\leftarrow SW0_1$
   - Once again, since there are no placed CBS, we start at p$\leftarrow 0$
   - Using again Equation 3.35, we compute the respective IdleSlope, $I^0_{SW0_1} = 34416827$ bps, still with $margin = 1$
7. **Lines 30–32**:
   - The directly affected flows (DA) are: $\{f_2, f_3\}$ which traverse the affected ports
   - The only indirectly affected flow (IA) is $\{f_4\}$ as it shares paths with both f_0 and f_2

- Thus, these are removed from the list: `unschedNonShaped.remove`($\{f_2, f_3, f_4\}$).

8. **Line 13**: Observe the list of unaffected flows (`unschedNonShaped`) is now empty, thus the algorithm returns the current configuration.

As discussed, this computed configuration is validated by the Schedulability Analyzer. Therefore, the approach was able to solve all schedulability issues by clustering deployed CBS to a single device.

3.4.5 Complexity Analysis Computation

The Burstiness-aware Partial CBS Deployment Algorithm, as detailed in Algorithm 1, operates with a time complexity dependent on the number of devices and flows in the network.

Theorem 3.1
The computational complexity of the Burstiness-aware Partial CBS Deployment Algorithm is:

$$O\left(|\mathcal{F}|^3 \cdot |DEV|^2 + |\mathcal{F}|^2 \cdot |DEV|^2 \cdot \log(|DEV|)\right)$$

where $|\mathcal{F}|$ is the number of flows and $|DEV|$ is the number of devices in the network.

Proof. The overall complexity of Algorithm 1 is determined by analyzing the cost of its primary operations. The analysis is broken down into the main components of the algorithm.

1. **Initial Reconfiguration:** The optional call to `recomputeIdleSlopes` recalculates the IdleSlope for all output ports across all devices. Inside this section of the algorithm, there are two main operations, the one on line 4, which has a complexity of $O(1)$ as it is a simple arithmetic operation, and the one on line 8, which calls the `recomputeIdleSlopes` function. This function iterates at most through all devices and their output ports to recalculate the IdleSlope values based on the new margin. The complexity for computing the IdleSlope for a single output port is $O(|\mathcal{F}|)$ due to the summation over all flows traversing that port. For every device, at most every output port is checked. There are at most $|DEV| - 1$ output ports in a device. Therefore, the complexity of `recomputeIdleSlopes` is the product between the square of the number of devices and the maximum number of flows traversing a port, leading to:

$$O(|DEV|^2 \cdot |\mathcal{F}|) \tag{3.38}$$

2. **Main Loop Analysis:** The algorithm's core is the `while` loop, which iterates as long as there are unscheduled non-shaped flows. In the worst-case scenario, only one flow is successfully scheduled and removed per iteration. Looking over each iteration, this results in $|F(p > 0)| + (|F(p > 0)| - 1) + (|F(p > 0)| - 2) + \cdots + 1$. This can be bounded

to $|\mathcal{F}|^2$, meaning the complexity of the loop itself is:

$$O(|\mathcal{F}|^2) \tag{3.39}$$

The complexity of the operations inside the loop consist of three main stages:

- **Flow Selection** First, we look at the function `MostConstrainedFlow` which selects a flow to service, this explores every unscheduled flow that doesn't belong to the highest priority. The expression can be bounded by $|\mathcal{F}|$, thus, for this function, we consider a complexity of
$$O(|\mathcal{F}|) \tag{3.40}$$
- **Device Selection and port ordering:** The *device selection* logic may, in the worst case, iterate through all devices, $|DEV|$. Then, for each device considered, the most demanding operation is sorting its output ports via `getOrderedOps`, for which we attribute a complexity $O(|DEV|\log(|DEV|))$ since it is the lower bound that can be attained by comparison-based sorting algorithms [101]. Therefore, the total complexity of finding a valid device for one flow and sorting its ports is:
$$O(|DEV| + |DEV|\log(|DEV|)) = O(|DEV|\log(|DEV|)) \tag{3.41}$$
- **CBS Placement and Configuration:** The inner `for` loop iterates over the sorted output ports of the selected device. In the worst case, it processes all output ports of the device, which can be bounded by $|DEV| - 1$ as one device might at most be connected to every other device. For each port, it computes the minimum IdleSlope using Equation 3.34, which involves summing over all flows traversing that port. This computation has a complexity of $O(|\mathcal{F}|)$. Therefore, the complexity of the inner loop is:
$$O(|DEV| \cdot |\mathcal{F}| - |\mathcal{F}|) = O(|DEV| \cdot |\mathcal{F}|) \tag{3.42}$$
The equivalence shown in Equation 3.42 stems from the fact $|DEV| \cdot |\mathcal{F}|$ dominates the $\mathcal{F}$ term in large-scale networks, allowing us to simplify the expression for clarity in complexity analysis.
The rest of the operations within the inner for loop are $O(1)$ operations, as they involve simple condition checks and assignments.
- **Influence of CBS placement:** After configuring a shaper, the algorithm identifies the set of Directly Affected (DA) and Indirectly Affected (IA) flows. The computation of DA stems from its very definition in Equation 3.31, which loops at worst over all unscheduled non-shaped flows traversing a device. This can be bounded by $|\mathcal{F}|$, leading to a complexity of $O(|\mathcal{F}|)$.
The IA computation, defined in Equation 3.32, is more costly. It involves checking each flow inside DA and the set of shaped flows to find intersections with the paths of the remaining unscheduled non-shaped flows. This operation involves checking each path of the flows within $DA \cup$ `shapedFlows`. This can be bounded by $|\mathcal{F}| \cdot |DEV|$. Thus leading to a complexity of $O(|\mathcal{F}| \cdot |DEV|)$ each time IA is

computed.

These two observations lead to a total complexity for updating the unscheduled flows list of:

$$O(|\mathcal{F}| + |\mathcal{F}| \cdot |DEV|) = O(|\mathcal{F}| \cdot |DEV|) \tag{3.43}$$

Considering this analysis, we can formalize the complexity of one `while` loop run:

$$O\Big(|\mathcal{F}|^2 \cdot (|\mathcal{F}| + |DEV| \cdot (|DEV| \cdot \log(|DEV|) + |DEV| \cdot |\mathcal{F}|) + |\mathcal{F}| \cdot |DEV|)\Big) \tag{3.44}$$

3. **Total Complexity:** To find the total complexity, we add Equations 3.38 and 3.44.

$$O\Big(|DEV|^2 \cdot |\mathcal{F}| + |\mathcal{F}|^2 \cdot \Big(|\mathcal{F}| + (|DEV| \cdot \log(|DEV|) + |DEV| \cdot |\mathcal{F}|) + |\mathcal{F}| \cdot |DEV|\Big)\Big)$$
$$= O\Big(|DEV|^2 \cdot |\mathcal{F}| + |\mathcal{F}|^3 + |\mathcal{F}|^2 \cdot |DEV| \cdot \log(|DEV|) + |\mathcal{F}|^3 \cdot |DEV|^2 + |\mathcal{F}|^3 \cdot |DEV|\Big)$$

Removing the lower order terms, we arrive at the final complexity expression:

$$O\Big(|\mathcal{F}|^3 \cdot |DEV|^2 + |\mathcal{F}|^2 \cdot |DEV| \cdot \log(|DEV|)\Big)$$

□

3.5 Framework Validation Over an Automotive Use-Case

To test the effectiveness of the Burstiness-aware Partial CBS Deployment Algorithm described in Section 3.4.4, we have applied it to an automotive network topology adapted from [102]. This topology is representative of a real-world industrial use case, featuring a mix of high-priority shaped flows and medium-priority non-shaped flows, as detailed in Table 3.6. The main objective of this validation is to assess the algorithm's impact on network costs and how it affects the performance of both shaped and non-shaped flows. This assessment is conducted by comparing the Partial CBS deployment against a Full CBS deployment, where we consider every port traversed by a high priority ($p = 0$) flow to be CBS-enabled. As such, as every switch is traversed by at least one high-priority flow, every switch is TSN-enabled, i.e., all ports are CBS-enabled. We also compare against a baseline scenario where no CBS is deployed, and only NP-SP is used to schedule traffic.

Note on Full CBS Implementation. To obtain a good Full CBS Implementation, we implemented a greedy search of the IdleSlope ensuring the minimum IdleSlope that can still schedule all high priority traffic. This is done to ensure that the Full CBS implementation is not over-provisioned, which would lead to an unfair comparison. As all devices are considered TSN-capable, we consider all the ports to have CBS deployed. We consider two subset of ports. First, ports with no high priority traffic. Those will use an IdleSlope equal to the port's bandwidth, essentially annuling CBS. Second, the output ports with at least one high-priority flow traversing them. Since we

consider in this case only three priorities and assume the highest priority flows are all schedulable under NP-SP scheduling, we only focus on ports being traversed by flows of priority $p = 0$. For this subset of ports, we decrement the value of the IdleSlope by a constant value until one of the high-priority flows becomes unschedulable. The last valid configuration is selected and the ports who affected the schedulability have the IdleSlope values frozen. We continue decrementing IdleSlopes until all ports have their IdleSlope values frozen.

3.5.1 Network Topology and Architectural Context

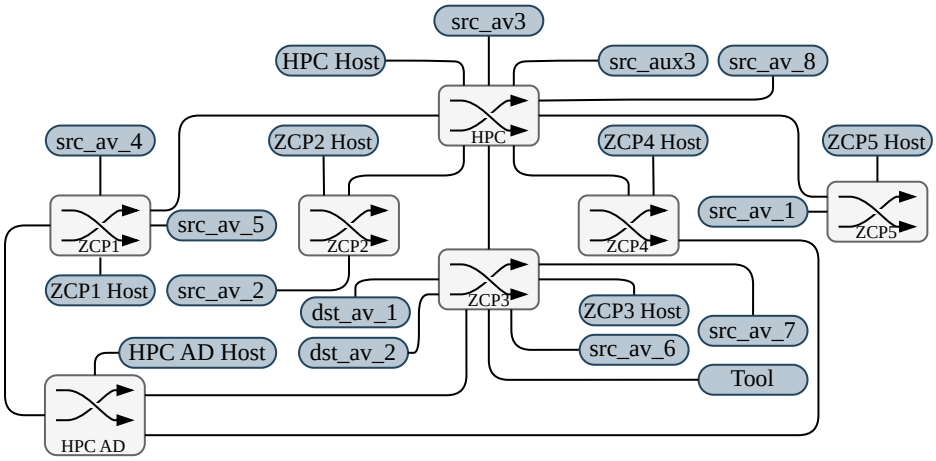


Figure 3.7: Adapted Industrial Automotive Topology from [102]

The network topology, depicted in Figure 3.7, represents a modern zonal automotive architecture adapted from [102]. The network comprises 7 Switches and 19 End Systems, organized to simulate a realistic in-vehicle backbone. The architecture distinguishes between two primary types of nodes:

- **Zone Control Processors (Zone Control Processor (ZCP)):** Represented as sources and destinations for control traffic, these nodes aggregate sensor and actuator data from specific zones of the vehicle.
- **High Performance Computing (High-Performance Computing (HPC)) Hosts:** These centralized units process high-bandwidth data for ADAS and infotainment.

3.5.2 Traffic Characteristics and Interference Patterns

The traffic profiles, detailed in Table 3.6, are heterogeneous where flows with conflicting requirements compete for bandwidth. The flows are categorized into three distinct priority classes, creating specific scheduling challenges:

Name	Priority	Rate (kB/s)	Deadline (ms)	Burst (B)	Sources	Destinations
AVB_1	0	2224	1.25	2780	src_av_1	dst_av_{1-2}
AVB_2	0	472	10	15570	src_av_{3-7}	HPC host
AVB_3	0	1307	33	43140	src_av_8	HPC host
CAN_1	1	613	0.075	46	ZCP1 Host, HPC Host	ZCP{1-5}, HPC Host
CAN_2	1	1040	0.075	78	ZCP1 Host, HPC Host	ZCP{1-5}, HPC Host
CAN_3	1	1146	0.075	86	ZCP1 Host, HPC Host	ZCP{1-5}, HPC Host
CAN_4	1	560	0.075	42	ZCP1 Host, HPC Host	ZCP{1-5}, HPC Host
FILE	2	63840	-	12768000	HPC Host	Tool

Table 3.6: Adapted Flow Characteristics for the Automotive use-case in [102]

1. **AVB Streams (Priority $p = 0$):** These flows (AVB_1 through AVB_3) represent multimedia or sensor data streams. They are characterized by large payload sizes and significant burstiness (up to 43,140 Bytes). While they have relatively loose deadlines (1.25 ms to 33 ms), their high burst size creates a risk of network congestion.

2. **Critical Control Traffic (CAN-to-Ethernet, Priority $p = 1$):** These flows (CAN_1 through CAN_4) encapsulate legacy CAN signals into Ethernet frames. In contrast to AVB, they have small payloads (42–86 Bytes) but require ultra-low latency, indicated by the extremely tight deadline of 0.075 ms.

3. **Best-Effort Traffic (Priority $p = 2$):** Represented by the FILE flow, this acts as background interference with high bandwidth but no strict timing constraints.

The primary challenge in this use case is the interference caused by the bursty AVB traffic on the latency-sensitive CAN traffic. Although the AVB flows operate at priority $p = 0$, their large bursts can saturate switch buffers and transmission links. Without intervention, a large AVB frame currently in transmission can delay a critical CAN frame long enough to violate its 0.075 ms deadline. Consequently, the AVB flows are identified as candidates for Credit Based Shaping (CBS). The goal of applying CBS here is to smooth out the AVB bursts, thereby preventing bandwidth monopolization and improving fairness for the critical CAN flows ($p = 1$).

3.5.3 Evaluation Scenarios

The evaluation is conducted in two distinct phases to analyze the trade-offs between hardware complexity and scheduling guarantees:

- **Partial vs. Full CBS Deployment:** First, we assess the impact of deploying CBS only on selected nodes (Partial) versus all nodes (Full). This comparison focuses on resource usage and configuration complexity, looking at the number of used TSN-enabled devices, and performance by looking at end-to-end delay bounds. The hypothesis is that Partial CBS can achieve similar performance guarantees to Full CBS but with reduced deployment costs.

- **Partial CBS vs. Non-Preemptive Static Priority (NP-SP):** Second, we compare the proposed Partial CBS solution against a baseline NP-SP scheduler. This phase aims to quantify the improvement in schedulability for the unshaped CAN flows.

The quantitative results for these comparisons, including cumulative distribution functions (CDFs) of the observed latencies, are detailed in next section.

3.5.4 Results and Analysis

Costs

In Table 3.7, we present the IdleSlope values computed by the Burstiness-aware Partial CBS Deployment Algorithm for the industrial use-case network that guarantee all the deadline constraints.

The results show that only 2 out of 7 devices require TSN-enabling, which represents a 70 % reduction in hardware costs compared to a Full CBS deployment. Furthermore, the algorithm instantiates only 3 out of 34 possible CBS shapers, achieving a 91 % reduction in configuration overhead.

This selective deployment demonstrates the algorithm's ability to strategically place CBS instances where they are most needed, leading to notable costs savings.

The obtained results for the industrial use-case highlight our heuristic algorithm's efficiency to effectively balance performance requirements, with cost savings through enabling legacy components use.

Table 3.7: Deployed CBS Parameters for the Industrial Use-Case Network

Device	Output Port (Link)	IdleSlope (Mbps)
HPC Switch	HPC → HPC Host	123.23
HPC Switch	HPC → ZCP3	55.98
ZCP3 Switch	ZCP3 → HPC	123.26

Performance and Schedulability

We analyze herein the impact of the different CBS deployments on flow latencies. Figures 3.8 and 3.9 present the Cumulative Distribution Functions (CDFs) of delay bounds for both shaped and unshaped flows, respectively.

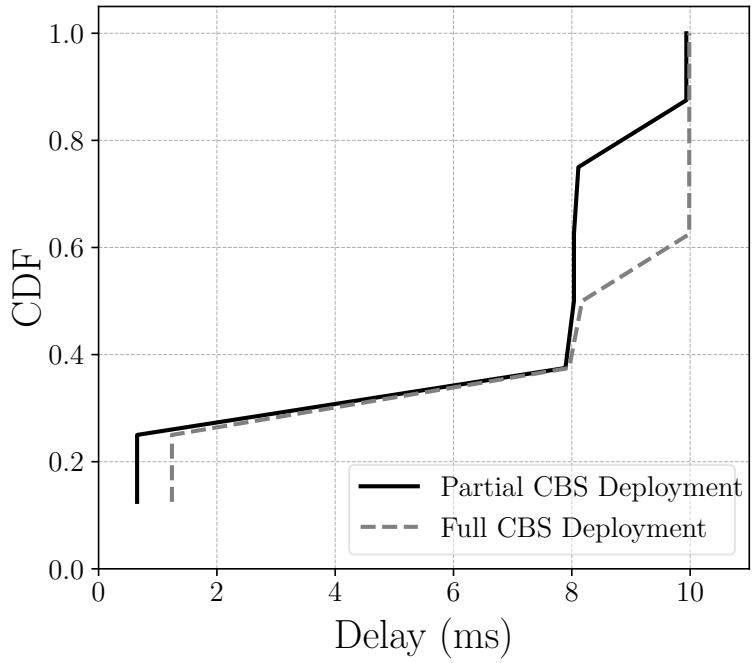


Figure 3.8: Shaped Flows ($p = 0$): Comparison of Full CBS vs. Partial CBS.

In Figure 3.8, we observe the behavior of the shaped AVB flows ($p = 0$) under Partial CBS and Full CBS deployments. The Partial CBS CDF consistently dominates the Full CBS CDF, indicating improved latency performance. For instance, we note that almost 80% of the flows have a maximum latency bound lower than 8 ms under Partial CBS, whereas only less than 50% of the flows achieve this bound under Full CBS. Globally, the Partial CBS yields a maximum delay bound reduction of 48 % compared to Full CBS.

This result demonstrates that the selective CBS placement under Partial CBS decreases the blocking effect suffered by shaped priority, leading to better performances for shaped priority in comparison to the Full CBS.

Figure 3.9 illustrates the behavior of unshaped control flows ($p = 1$) under NP-SP and Partial CBS deployment. As it can be noticed, under the standard NP-SP scheduler (No CBS), 60 % of these flows violate their deadlines due to interference from bursty AVB traffic.

On the other hand, all priority $p = 1$ flows meet their deadlines, ensuring 100 % schedulability under Partial CBS deployment. This is due to the improvement of the service fairness under CBS deployment. Indeed, CBS enables smoothing the AVB bursts, which leaves more

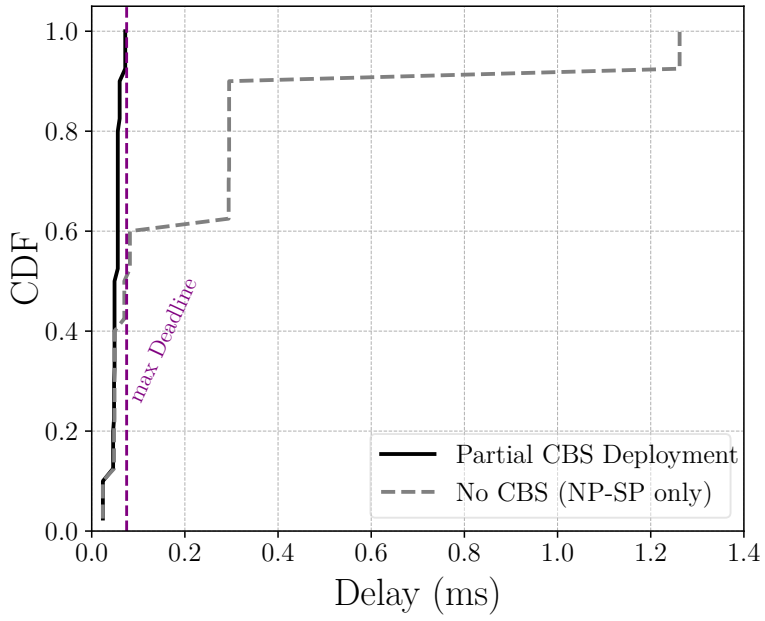


Figure 3.9: Unshaped Flows ($p = 1$): Comparison of Partial CBS vs. NP-SP (No CBS).

room to serve CAN flows, and consequently improving their delay bounds.

Unlike NP-SP, Partial CBS deployment has shown its ability to solve all the schedulability issues of CAN flows, while minimizing the number of CBS with reference to Full CBS.

3.6 Conclusions

In this chapter, we have explored the feasibility of a transition-friendly approach to TSN. We proposed a framework for the *partial* deployment of CBS, aiming to answer the research questions defined in the introduction: *where* to place CBS, *when* they are beneficial, and *how* to configure them.

Through the *Burstiness-aware Partial CBS Deployment* heuristic, we have provided key elements to answer these questions:

- Regarding **R.Q.1 (Where)**, the investigation suggests that effectiveness lies in targeting the boundaries where high-priority traffic interferes with non-schedulable flows, rather than ubiquitous deployment.
- Regarding **R.Q.2 (When)**, the results indicate that when NP-SP fails to guarantee the schedulability requirements of medium/low priorities traffic, partial CBS deployment would be a viable strategy to guarantee all the schedulability requirements, while offering better performance for shaped traffic and lower costs in comparison to full deployment.

- Regarding **R.Q.3 (How)**, the analysis points towards a configuration strategy, minimizing the IdleSlope to maintain schedulability of shaped traffic without starving lower-priority traffic.

The validation of the framework on an industrial use-case has shown that Partial CBS deployment is cost-effective, while guaranteeing performance requirements for all traffic classes.

However, we have considered in this chapter CBS, deployed as an isolated TSN mechanism and without taking into account its interaction with other TSN mechanisms, and particularly those to guarantee reliability like Frame Replication and Elimination for Redundancy (FRER).

This challenge will be handled in the next chapter, where we will refine our proposed CBS deployment strategy to be *FRER-aware*.

Chapter 4

FRER-Aware Partial CBS Deployment

Contents

4.1 Extended System Model and FRER Background 64
4.1.1 FRER Operational Principles 65
4.1.2 Extended Network Model and Architecture 66
4.1.3 Extended Traffic Model 66
4.1.4 NC model for FRER 67
4.2 Motivation for FRER-aware Partial CBS Deployment 68
4.2.1 The Impact of FRER on Traffic Burstiness 68
4.2.2 Understanding CBS impact in FRER-enabled networks 70
4.3 FRER-aware Partial CBS Deployment as an Optimization Problem . 71
4.3.1 Problem Linearization and Relaxation 71
4.4 Framework for FRER-aware Partial CBS Deployment 75
4.4.1 Modifications to the Existing Framework 75
4.4.2 FRER-aware CBS Placement 76
4.4.3 FRER-aware CBS Configuration 76
4.4.4 Adapting the Existing Partial CBS Deployment Algorithm for FRER-Awareness 78
4.4.5 Complexity Analysis Computation 82
4.5 Validation over an Automotive Use-Case 83
4.5.1 Network Topology and Traffic Model 83
4.5.2 Results and Analysis 85
4.6 Conclusions 87

In the previous chapter, we presented a framework for the partial CBS deployment in TSN networks. We have shown that by strategically placing and configuring CBS instances, it is possible to mitigate the traffic burstiness inherent to the traffic profile and increased when crossing legacy switches. This mitigation enables balancing cost-efficiency with performance guarantees. However, the proposed framework operates under the assumption that CBS is used on its own and does not account for burstiness increase due to reliability mechanisms.

In safety-critical networks, mechanisms such as FRER are essential. Yet, the literature indicates that the elimination function of FRER can substantially increase burstiness [92], does not efficiently account for or mitigate. Thus, our previous heuristic is denoted in this chapter as *FRER-blind* heuristic.

In this chapter, we extend the presented framework in Chapter 3 by introducing a FRER-aware partial CBS deployment heuristic. This extension allows us to refine our answers to the research questions defined in this dissertation, specifically addressing **R.Q.1**: *Where can CBS be effectively deployed to leverage its benefits, while maintaining interoperability with legacy Ethernet devices?* and **R.Q.3**: *How can CBS be configured to obtain the best performance trade-off in a partial deployment scenario when interacting with other TSN mechanisms?*

Our motivation for this extended approach is two-fold. First, we aim to ensure that the use of FRER does not compromise the deterministic latency guarantees of the network due to increased burstiness. Second, we seek to maintain the cost-effectiveness of the partial deployment strategy by only deploying CBS where it is strictly necessary to manage the extra interferences due to FRER. Consistent with our previous framework, we utilize the NC framework to model the interaction between FRER and CBS.

This chapter is organized as follows:

- An extended system model in Section 4.1, detailing the FRER operation and its NC modeling, is provided.
- The problem motivation in Section 4.2, analyzing the specific impact of FRER on traffic burstiness and why the previous FRER-blind partial CBS deployment is insufficient is defined.
- A formulation of the FRER-aware Partial CBS Deployment is provided as an optimization problem in Section 4.3.
- The extended framework and the modified heuristic algorithm to solve this problem in Section 4.4 are presented.
- A performance evaluation of the FRER-aware framework over the already detailed automotive use-case in Section 4.5 is provided.

4.1 Extended System Model and FRER Background

In this section, we update the system model detailed in Section 3.1 to support the interaction between CBS and FRER, as well as the redundancy concepts at the traffic and network levels under FRER. We first detail the operational principles of FRER. We then extend the network and the traffic models to account for redundant disjoint paths inherent to FRER mechanism. Finally, we present the NC model used to formally analyze the impact of FRER on traffic burstiness.

4.1.1 FRER Operational Principles

The FRER mechanism, standardized in IEEE 802.1CB, ensures seamless network redundancy by distributing stream content across independent paths. This mechanism relies on a fundamental distinction between *Member Streams*, the individual redundant flows traversing disjoint paths, and the *Compound Stream*, which represents the logical aggregation of all member streams. Figure 4.1 illustrates the distinction between these two stream types. The operation of FRER is driven by the following two primary functions:

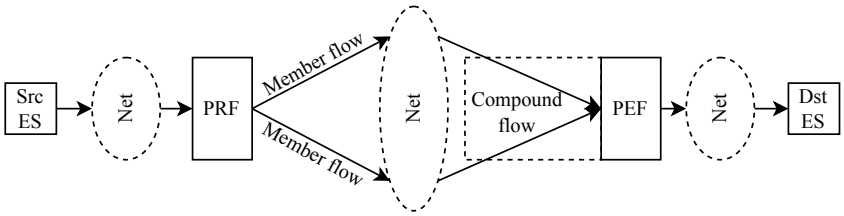


Figure 4.1: Architectural Overview of FRER operations

Frame Replication Function The replication process is governed by the *Packet Replication Function (PRF)*. This function accepts a single packet from the upper layers and generates multiple replicas, transmitting each via a separate, disjoint path. Consequently, this function effectively expands a single input flow into the necessary set of Member Streams required to achieve the required reliability level.

Frame Elimination Function Conversely, the reconstruction of the original stream is performed by the *Packet Elimination Function (PEF)*, specifically through its internal *Sequence Recovery Function (SRF)*. The SRF uses sequence numbers assigned at the source to filter incoming Member Streams and discard duplicates. The standard delineates two specific algorithms for this purpose:

- **Match Recovery Algorithm (MRA):** Designed for *intermittent streams*, this algorithm accepts the first valid packet and discards subsequent matches. Its correct operation requires that the latency difference between paths results in an overlap of at most one packet. This algorithm is adequate for periodic non-bursty streams.
- **Vector Recovery Algorithm (VRA):** Tailored for *bulk streams*, this algorithm maintains a history window (vector) of received sequence numbers. It accepts packets falling within a valid range while rejecting those outside of it or those previously received. This algorithm handles bursty streams, and it is the one commonly used in TSN networks.

The architectural integration of these functions is illustrated in Figure 4.1, which visualizes the propagation of Member Streams across disjoint paths and their subsequent convergence and filtering at the destination.

4.1.2 Extended Network Model and Architecture

We extend the network model presented in Section 3.1 to incorporate FRER functionality. As stated, burstiness increase happens after elimination functions. Thus, we need to model these within the network architecture. To do so, we consider the Output Port from Figure 3.1 to be extended with potential elimination functions as shown in Figure 4.2.

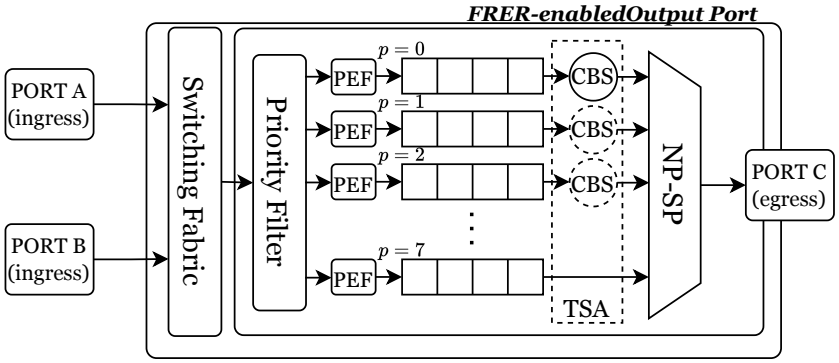


Figure 4.2: Detailed Switch Architecture with FRER Elimination Function

To model this potential addition within our system model, we use a binary parameter PEF_{op}^{p}, which considers PEF instances for each priority-queue of *op*. This parameter indicates whether the Output Port *op* includes a FRER elimination function for class p: $\text{PEF}_{op}^{p} = 1$ indicates elimination, while $(\text{PEF}_{op}^{p} = 0)$ indicates no elimination for (op, p).

4.1.3 Extended Traffic Model

To extend the traffic model defined in Section 3.1, we must account for the presence of redundant paths introduced by FRER. Their introduction creates a distinction between an abstract flow and its network realization. An abstract flow f is realized by one or more *instances*, contained in the set $\mathcal{I}^f$.

- **Without FRER:** The flow is not replicated. Its set of instances contains a single element, $\mathcal{I}^f = \{f\}$, which follows a single, pre-defined path ϕ^f.
- **With FRER:** The flow is replicated for redundancy. The abstract flow f acts as a *compound flow*, which is split into multiple *member flows* (instances), $\mathcal{I}^f = \{f_0, f_1, \ldots, f_k\}$, with $k > 0$. Each member flow f_i is routed along its own path, ϕ^{f_i}, though they share the same source and destination.

Each flow within $\mathcal{I}^f$ is characterized with the same set of specifications defined for the abstract flow f, $(r^f, b^f, L^f, D^f, p^f)$ for rate, burst, packet-length, deadline and priority, respectively.

Thus, we update Table 3.1 from Section 3.1 to include the extended notations introduced for FRER in Table 4.1.

Table 4.1: Summary of notations

General Network, CBS and FRER Notations	
OP, E	Set of output ports, Set of unidirectional links
op	An output port $\in OP$
C	Transmission capacity of a link
DEV	Set of devices ($SW \cup ES$)
SW, ES	Sets of switches and end systems
$op.dev$	Device associated with port op
$OP(dev)$	Set of output ports of device dev
P	Set of priorities $\{0, \ldots, 7\}$
CBS_{op}^{p}	Binary: CBS active at op for priority p
PEF_{op}^{p}	Binary: PEF active at op for priority p
I_{op}^{p}	IdleSlope for CBS at op for priority p
$L_{op}^{>p,\max}$	Max. frame size of interfering traffic at op with lower priority than p
Flows Notations	
$\mathcal{F}$	Set of all abstract flow specifications
(r^f, b^f)	Source rate and burst size of f
L^f	Maximum frame size of f
D^f	End-to-end deadline of f
p^f	Assigned priority of f
$\mathcal{I}^f$	Set of flow instances using the same tuple template
ϕ^f	Path of flow f
d^f	End-to-end delay bound of f
$\mathcal{F}(p)$	Set of abstract flow specifications $f \in \mathcal{F}$ with priority p
$\mathcal{F}(op, p)$	Abstract flow specifications $f \in \mathcal{F}(p)$ traversing op
$\mathcal{F}(dev)$	Abstract flow specifications $f \in \mathcal{F}$ traversing dev

4.1.4 NC model for FRER

In [92], the authors show the burstiness increase induced after PEF when a compound flow is merged. This increase in burstiness has been analyzed in the general case of FRER operations, and integrated at the output arrival curves computation after a PEF. In our study, we consider a common implementation of FRER, where a single pair of PRF and PEF are applied for a given compound flow along its path from the source until the final destination. In this particular case, we apply Corollary 1 in [92] that is reproduced below for clarity using this dissertation's notations.

Corollary 4.1
Consider a compound flow f with an Arrival Curve $\alpha^f(t)$ that is replicated within N instances in $\mathcal{I}^f$ and sent through N disjoint paths $\{\phi^i\}_{i\in[1,N]}$ to be then merged at a single elimination point. Assume the delay along ϕ^i is bounded within $[d^i_{\min}, d^i_{\max}]$.

Then, the output Arrival Curve of the flow after the PEF is given by[1]*:*

$$\alpha_{PEF}^{f,*}(t) = \sum_{i=1}^{N} \alpha^{f_i}\left(t + \left(d^i_{\max} - d^i_{\min}\right)\right) \otimes \alpha^{f}\left(t + \left(\max_{i\in[1,N]} d^i_{\max} - \min_{j\in[1,N]} d^j_{min}\right)\right) \tag{4.1}$$

From Corollary 4.1, we can see that the output arrival curve of a compound flow f after the elimination function is bounded by the minimum of two terms. The first term is the sum of the arrival curves of all replicated instances in $\mathcal{I}^f$, each shifted by the difference between its maximum and minimum delay along its path. The second term is the original arrival curve shifted by the difference between the maximum of all maximum delays and the minimum of all minimum delays across all paths. This indicates that variations in delay across replicate paths directly increase the burstiness of the output traffic after a PEF.

4.2 Motivation for FRER-aware Partial CBS Deployment

In this section, we provide a toy example with two clear objectives: first, to illustrate the impact of frame replication on network performance with no CBS deployed (NP-SP only); second, to compare different CBS deployment strategies under the frame replication scenario. This example includes a first hypothesis on how to strategically place CBS in a FRER-aware manner. The NC analysis method used to compute delay bounds is FP-TFA++, detailed in Section A.6.4.

4.2.1 The Impact of FRER on Traffic Burstiness

Consider output port architecture depicted in Figure 4.3, where the output arrival curve after the elimination function is given by the sum of the arrival curves of all individual flow instances, i.e., $\sum_f \alpha_{PEF}^{f,*}$. As this is the arrival curve that enters the potential CBS to be deployed, the burstiness increase directly impacts the CBS configuration, particularly the IdleSlope value required to ensure flow schedulability. In addition, this burstiness increase given by the elimination function can be potentially treated with CBS, which might impact the way devices are selected for CBS placement. This motivates the need for a FRER-aware approach to CBS deployment that takes into account the increased burstiness due to frame replication and elimination.

To illustrate the impact of FRER on CBS deployment, we evaluate a baseline network topology of a TSN network shown in Figure 4.4, where three switches (SW0, SW1, and SW2) interconnect four end-systems (ES0, ES1, ES2, and ES3). The links between devices are full-duplex with a capacity of 1 Gbps. The network is traversed by four flows with characteristics summarized in Table 4.2.

[1]Where $\otimes$ corresponds to the min-plus convolution defined in Section A.1.1: $(f \otimes g)(t) = \inf_{0\leq s\leq t}\{f(s) + g(t-s)\}$

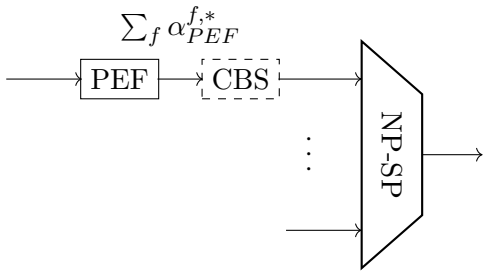


Figure 4.3: Output arrival curve after the elimination function

Notably, flow f_0 is duplicated using FRER and has two paths, while the other flows have a single path. The paths for f_0 are disjoint, ensuring that the replicated frames have independent paths.

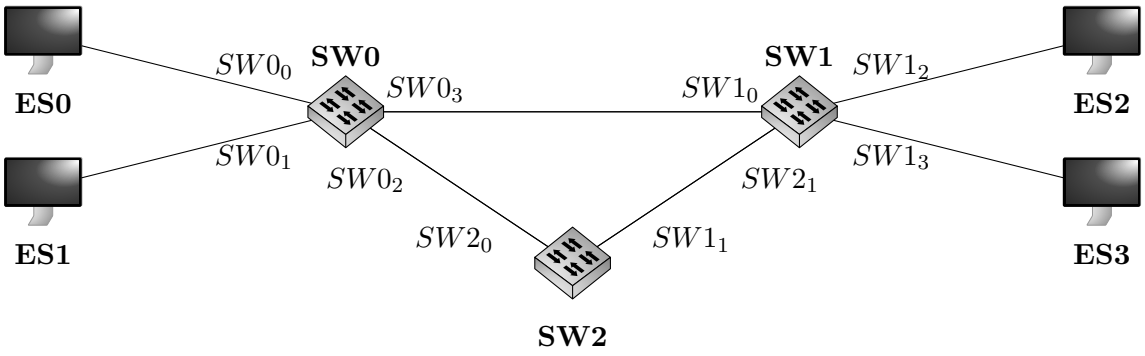


Figure 4.4: Example of a TSN Network with FRER

Table 4.2: Flow characteristics for the motivational example network (Figure 4.4).

Flow	**L^{max} (B)**	**b (# of frames)**	**p**	**Period (ms)**	**D (ms)**	ϕ^f
f_0	1000	100	0	5	5	{ES0,SW0,SW1,ES2}, {ES0,SW0,SW1,SW2,ES2}
f_1	500	1	1	2	2	{ES1,SW0,SW1,ES2}
f_2	500	1	1	3	3	{ES1,SW0,SW2,SW1,ES2}
f_3	1500	500	2	100	—	{ES3,SW1,ES2}

To show the impact of frame replication over the network performance, we compare in Table 4.3 flow delay bounds computed with NC for both scenarios with and without flow replication under NP-SP scheduling. The table shows that without frame replication only f_1 is unschedulable since its delay bound is higher than its deadline of 2ms. However, with frame replication, f_2 also becomes also unschedulable. This shows how the added burstiness created by the elimination function can lead to extra schedulability issues.

Table 4.3: Impact of FRER on delay bounds

Flow	Without frame replication	With frame replication
d^{f_0} (ms)	0.88	0.93
d^{f_1} (ms)	2.08	2.10
d^{f_2} (ms)	1.11	3.13
d^{f_3} (ms)	8.53	8.55

4.2.2 Understanding CBS impact in FRER-enabled networks

Having demonstrated the impact of FRER-induced burstiness on flow schedulability, we propose resolving this issue through targeted CBS deployment. We compare the different deployment approaches in terms of schedulability and number of deployed CBS, with reference to the NP-SP case. This will help us to better understand how we can adapt the burstiness-aware partial deployment approach from Chapter 3 to FRER-enabled networks. The computed delay bounds are summarized in Table 4.4.

The first deployment heuristic is the burstiness-aware partial CBS deployment, here called FRER-blind for comparison. Recall that with this approach, CBS deployment is done on the first device where the unscheduled flow with the highest difference between its delay and deadline encounters high-priority traffic. In contrast, the labeled FRER-aware partial CBS deployment places CBS at the output of the elimination function ($SW1_2, p = 0$), attempting to remove burstiness that causes additional schedulability issues. The result for both approaches is obtained with the smallest IdleSlope that allows for f_0 to be schedulable[2]. Full CBS Deployment uses the approach described in Note 3.5 in Chapter 3, where CBS is deployed on every output port of every switch for the highest priority-level.

For every CBS deployment strategy, all flows remain schedulable. However, to fix the non-schedulability issues, we only need one CBS under FRER-aware CBS deployment compared to 2 and 5 CBS under FRER-blind partial CBS deployment and full CBS deployment respectively.

Thus, through this simple example, we understand the impact of frame replication and elimination on traffic burstiness and network performance, and also see how CBS deployment strategies can be adapted to account for these effects. Strategically placing CBS at the output of the elimination function can achieve schedulability with lower deployment cost in FRER-enabled scenarios. This fact motivates the introduction of a FRER-aware partial CBS deployment approach in time-sensitive networks.

[2]We will explore IdleSlope selection later on in this chapter. The main challenge is to include the Output Arrival Curve from Eq. 4.1. For now, we will use the empirically found lowest possible IdleSlope that follows placement rules from either approach.

Table 4.4: Impact of CBS deployment approaches on delay bounds

Metric	No CBS (NP-SP)	FRER-blind Partial CBS Deployment	FRER-aware Partial CBS Deployment	Full CBS Deployment
IdleSlope(s) (Mbps)	—	$SW0_2$: 200 $SW0_3$: 199	$SW1_2$: 199	$SW0_2$: 385 $SW0_3$: 381 $SW2_1$: 385 $SW1_2$: 385 Others: 1000
d^{f_0} (ms)	0.93	4.99	4.99	4.99
d^{f_1} (ms)	2.10	1.11	1.08	0.16
d^{f_2} (ms)	3.13	2.14	2.09	0.20
d^{f_3} (ms)	8.55	8.55	7.89	10.2

4.3 FRER-aware Partial CBS Deployment as an Optimization Problem

The formulation of the main optimization problem for FRER-aware partial CBS deployment builds upon the same problem formulation in Chapter 3, but now integrates considerations for FRER-enabled networks.

The problem we aim to solve is still a placement problem, with the motivation of minimizing the number of TSN-enabled devices to improve legacy Ethernet compatibility. The solution to this problem shall respect the IdleSlope bounds from Equations 3.12 and 3.14. Flow deadlines must be respected and the order in which CBS are placed must follow the one set in Equation 3.16. Thus, the formulation of the original problem from Chapter 3 still holds. The key difference lies in the implicit delay bound computation required for constraint 3.15, where we now must account for the effects of the elimination function with Corollary 4.1 to correctly compute these bounds.

In addition, as the problem formulation is the same on its surface, the challenges from Section 3.3 remain. The problem is nonlinear and depends on recursive computations due to the use of FP-TFA++ to verify constraint 3.15. Thus motivating a linearization and relaxation of the problem accounting for FRER implementation.

4.3.1 Problem Linearization and Relaxation

The linearization of the objective function done in Equation 3.17 holds as it does not depend on FRER implementation.

However, the relaxation of the delay constraint into local delays from Equation 3.19 must be updated to account for flow replication and elimination.

Here, the idea remains the same as Equation 3.19, where the recursiveness from the FP-TFA++ algorithm is removed, by ignoring burst propagation. Hence, to remove such a recursive form, the constraint is relaxed by ignoring the burst propagation. The initial burst size of flows generated by the source to compute the minimum upper delay bound for each traffic class with priority p within each output port op is considered. Thus, $\underline{d}_{op}^{p}$ as follows:

$$\underline{d}_{op}^{p} = \begin{cases} \frac{\sum_{\forall f \in \mathcal{F}(op,p)} b^f}{R_{op}^{p}} + T_{op}^{p} & \text{if } \mathrm{PEF}_{op}^{p} = 0 \\ \frac{\sum_{\forall f \in \mathcal{F}(op,p)} (b^f + r^f \cdot J_{diff}^{f})}{R_{op}^{p}} + T_{op}^{p} & \text{if } \mathrm{PEF}_{op}^{p} = 1 \end{cases} \tag{4.2}$$

Where,

- R_{op}^{p} and T_{op}^{p} are computed using Theorem A.5

$$R_{op}^{p} = C - \sum_{p=0}^{p-1} r_{op}^{p}$$

$$T_{op}^{p} = \frac{\sum_{p'=0}^{p-1} b_{op}^{p'} + L_{op}^{>p,max}}{C - \sum_{p'=0}^{p-1} r_{op}^{p'}}$$

  or Equation 3.1

$$R_{op}^{p} = I_{op}^{p}$$

$$T_{op}^{p} = I_{op}^{p} \cdot \frac{\sum_{k=0}^{p-1} cred_{op}^{k,\min} - L_{op}^{>p,\max}}{\sum_{k=0}^{p-1} I_{op}^{k} - C}$$

  Where $cred_{op}^{p,\min}$ is the credit lower bound:

$$cred_{op}^{p,\min} = (I_{op}^{p} - C) \cdot \frac{L^{p}}{C},$$

  depending on whether CBS is implemented for the (op, p) of interest;

- if FRER is enabled for the (op, p) of interest, i.e., $\mathrm{PEF}_{op}^{p} = 1$, we consider the right term of the output Arrival Curve in Corollary 4.1 while ignoring burst propagation;

- J_{diff}^{f} is the maximum jitter among the different replicates paths of the compound flow f and it is computed considering that $\forall f_i \in \mathcal{I}^f$ the end-to-end delay along ϕ^{f_i} is bounded within $[d_{min}^{f_i}, d_{max}^{f_i}]$:

$$J_{diff}^{f} = \left(\max_{f_i \in \mathcal{I}_f} d_{\max}^{f_i} - \min_{f_j \in \mathcal{I}_f} d_{min}^{f_j} \right) \tag{4.3}$$

To relax Constraint 3.15, we must define a local deadline. This definition ensures that the sum of all local deadlines never exceeds the global deadline for a given flow. Our approach builds on Equation 3.20, where deadline proportions are assigned based on the port and

priority with the highest arrival rate. This heuristic assigns less constrained local deadlines for the busiest ports. By incorporating flow replication into this logic, we define the local deadline for output port op and priority p as follows:

$$D_{op}^{p} = \min_{\forall f \in \mathcal{F}(op,p)} \left\{ \min_{\forall f_i \in \mathcal{I}^f | op \in \phi^{f_i}} \left[\left(D^f - d_{src}^{p} - d_{prop} \right) \cdot \frac{r_{op}^{p}}{\sum_{op' \in \phi^{f_i} \setminus src} r_{op'}^{p}} \right] \right\} \tag{4.4}$$

Proof. The first steps of the proof are similar to the one of Equation 3.20. Here, the main difference is the second *min* of Equation 4.4, which we justify in this proof.

If $\text{PEF}_{op}^{p} = 1$, there exists a flow f for which multiple instances exist. Thus, we need to consider all instances $f_i \in \mathcal{I}^f$ such that $op \in \phi^{f_i}$ into equation 3.21. To do so, we use the instance of flow f_i that minimizes the $D^{f_i} - d_{src}^{p} - d_{prop}$ for all instances $f_i \in \mathcal{I}^f$ when computing the right hand side of Equation 3.21, when considering flow f.

To distribute the local deadlines among all output ports in the path of any flow $f \in \mathcal{F}(op, p)$ (excluding the source), we use a similar approach as in Equation 3.21. Here the local deadline for any output port op and priority p is proportional to the rate of flow f at output port op over the cumulative rate of flow f at all output ports in its path (excluding the source). Consider D_{op}^{p} as:

$$D_{op}^{p} = \min_{\forall f \in \mathcal{F}(op,p)} \left\{ \min_{\forall f_i \in \mathcal{I}^f | op \in \phi^{f_i}} \left[\left(D^f - d_{src}^{p} - d_{prop} \right) \cdot \frac{r_{op}^{p}}{\sum_{op' \in \phi^{f_i} \setminus src} r_{op'}^{p}} \right] \right\}.$$

Summing D_{op}^{p} over all output ports in the path of any flow $f \in \mathcal{F}(op, p)$ (excluding the source) verifies:

$$\sum_{\forall op \in \phi^f \setminus src} d_{op}^{p} \leq \sum_{\forall op \in \phi^f \setminus src} D_{op}^{p}.$$

This completes the proof. □

From Equations 4.2 and 4.4, we may rewrite Equation 3.19 with the extended FRER-aware

definitions as follows:

$$D^p_{op} = \min_{\forall f \in \mathcal{F}(op,p)} \left\{ \min_{\substack{\forall f_i \in \mathcal{I}^f \\ op \in \phi^{f_i}}} \left[(D^f - d^p_{src} - d_{prop}) \cdot \frac{r^p_{op}}{\sum_{op' \in \phi^{f_i} \setminus \{src\}} r^p_{op'}} \right] \right\}$$

$$\underline{d}^p_{op} = T^p_{op} + \frac{1}{R^p_{op}} \cdot \begin{cases} \sum\limits_{\forall f \in \mathcal{F}(op,p)} b^f & \text{if } \mathrm{PEF}^p_{op} = 0 \\ \sum\limits_{\forall f \in \mathcal{F}(op,p)} (b^f + r^f \cdot J^f_{diff}) & \text{if } \mathrm{PEF}^p_{op} = 1 \end{cases}$$

$$\therefore \quad \underline{d}^p_{op} \leq D^p_{op} \cdot \mathrm{margin} \tag{4.5}$$

Here, *margin* serves the same purpose as in Equation 3.19, compensating for the relaxation made by ignoring burst propagation.

The relaxation considerations above lead to the consolidation of the linear optimization sub-problem, which is now valid for both FRER-enabled and FRER-disabled TSN networks. The formulation is similar to the one in Section 3.3.2, but with the updated definition of constraint 3.27. The complete formulation of the linear optimization sub-problem for FRER-aware partial CBS deployment is as follows:

Objective

$$\text{minimize} \quad \sum_{dev \in DEV} TSN_{dev} \tag{4.6}$$

Subject to

$$\sum_{p \in P} I^p_{op} \cdot CBS^p_{op} \leq 0.75 \cdot C, \qquad \forall op \in OP \tag{4.7}$$

$$I^p_{op} \geq r^p \cdot CBS^p_{op}, \qquad \forall op \in OP, \forall p \in P \tag{4.8}$$

$$TSN_{dev} \geq CBS^p_{op}, \qquad \forall dev \in DEV, \; \forall op \in OP(dev), \forall p \in P \tag{4.9}$$

$$\underline{d}^p_{op} \leq D^p_{op}, \qquad \forall op \in OP, \forall p \in P \tag{4.10}$$

$$CBS^p_{op} \in \{0, 1\}, \qquad \forall op \in OP, \forall p \in P \tag{4.11}$$

$$TSN_{dev} \in \{0, 1\}, \qquad \forall dev \in DEV \tag{4.12}$$

$$CBS^p_{op} \geq CBS^{p+1}_{op}, \qquad \forall op \in OP, \forall p \in P \tag{4.13}$$

Where Constraints 4.9, 4.11 and 4.12 correspond to the linearization of the objective function 3.16 in the original problem. Constraints 4.7 and 4.8 ensure that the IdleSlope values respect the bounds defined by the network capacity and the cumulative rate of flows.

Constraint 4.10 enforces the local schedulability condition derived in Equation 4.5. Finally, Constraint 4.13 maintains the placement order condition for CBS deployment across priority levels.

4.4 Framework for FRER-aware Partial CBS Deployment

4.4.1 Modifications to the Existing Framework

The FRER-aware Partial CBS Deployment Framework extends the framework presented in Chapter 3, by incorporating the effects of FRER. While the high-level architecture remains consistent, as depicted in Figure 4.5, our contribution lies in adapting the core logic to account for FRER's impact on network traffic within the *FRER-Aware Partial CBS Deployment* module. Thus CBS placement and configuration now considers FRER's presence by changing placement strategy and the computation of the IdleSlope. Moreover, the NC model now incorporates Corollary 4.1 to correctly compute delay bounds under FRER-enabled conditions.

The framework still revolves around two main components: the *FRER-aware Partial CBS Deployment* algorithm and the *FRER-Aware Schedulability Analysis* modules. The first component is tasked with strategically placing and configuring CBS, whereas the second validates the network's timing constraints using NC.

A detailed description of the additions to the two main components is provided below:

FRER-aware Partial CBS Deployment The key enhancement here is the integration of FRER-specific behaviors into this process. Frame elimination, for instance, can increase traffic burstiness at the nodes where redundant copies are discarded as seen in the previous sections of this chapter. Our new approach addresses this by prioritizing the placement of CBS at these elimination nodes to continue reducing burstiness where it most matters. Furthermore, since the arrival curve now includes new components related to FRER, the formula for calculating the minimum IdleSlope (Equation 3.35) must be updated accordingly.

Schedulability Analysis This component acts as the verification engine of our framework. It employs NC theory, specifically the state-of-the-art FP-TFA++ algorithm, to compute worst-case end-to-end delay bounds for every flow in the network. The analysis is fully FRER-aware, modeling the effects of both frame replication and elimination. It takes a complete network specification as input, either the initial design or a modified one with our proposed CBS placements, and outputs the calculated delay bounds. These bounds are then checked against the flows' deadlines to formally verify the schedulability.

In the next section, we detail the heuristic algorithm developed for the *FRER-aware Partial CBS Deployment* component. To this end, we will formalize the FRER-aware placement

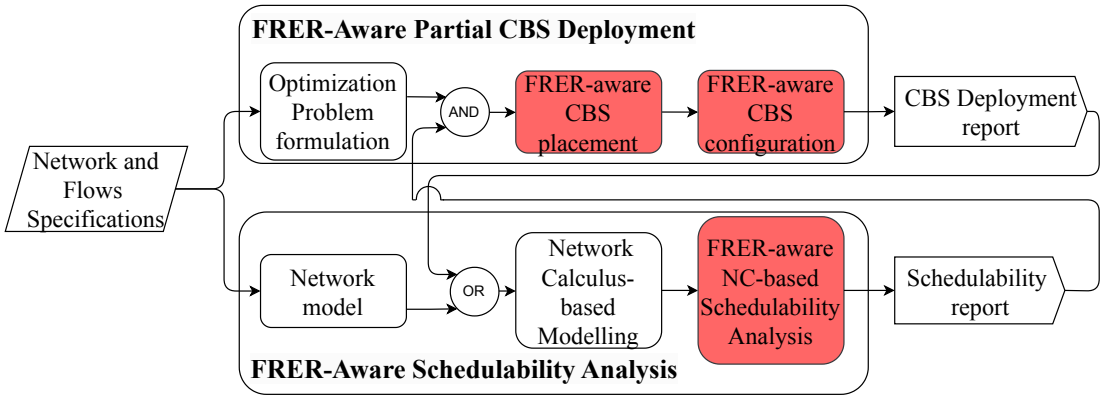


Figure 4.5: Overview of the FRER-aware Partial CBS Deployment Framework

strategy and the IdleSlope configuration method from Figure 4.5 before formalizing the complete algorithm.

4.4.2 FRER-aware CBS Placement

Our heuristic for CBS placement places CBS instances iteratively based on the process described on Figure 4.6. The goal is to obtain a device for which to perform CBS placement. The strategy is almost the same as the one from Section 3.4.2 with one key difference. Figure 4.6 shows that after selecting the foi, instead of directly selecting the first device where it contends with higher-priority traffic, we first check if any device along its path implements a PEF. If such a device exists, we select it as the target for CBS placement. This change is motivated by the burstiness increase caused by frame elimination at PEF nodes, which can lead to additional schedulability issues as shown in Section 4.2. By placing CBS at these nodes first, we aim to mitigate the burstiness impact before addressing other unscheduled flows.

4.4.3 FRER-aware CBS Configuration

To place pertinent CBS instances, we must also configure their Idleslopes to favor the schedulability of all flows in the network. As discussed in Chapter 3.4.3, this corresponds to the lowest possible value that still guarantees the schedulability of all shaped flows. However, with FRER, the output arrival curve at the elimination function injected into the CBS includes an added burstiness that was not considered in Equation 3.35. Thus, we must update the derivation of the minimum IdleSlope to account for this burstiness increase, while still being compatible with the original derivation when no FRER is implemented.

Starting from Equation 4.5 and considering a placed CBS at (op, p) where a PEF is implemented, i.e., $\mathrm{CBS}^p_{op} = 1, \mathrm{PEF}^p_{op} = 1$, and considering R^p_{op} and T^p_{op} from Equation 3.1 the

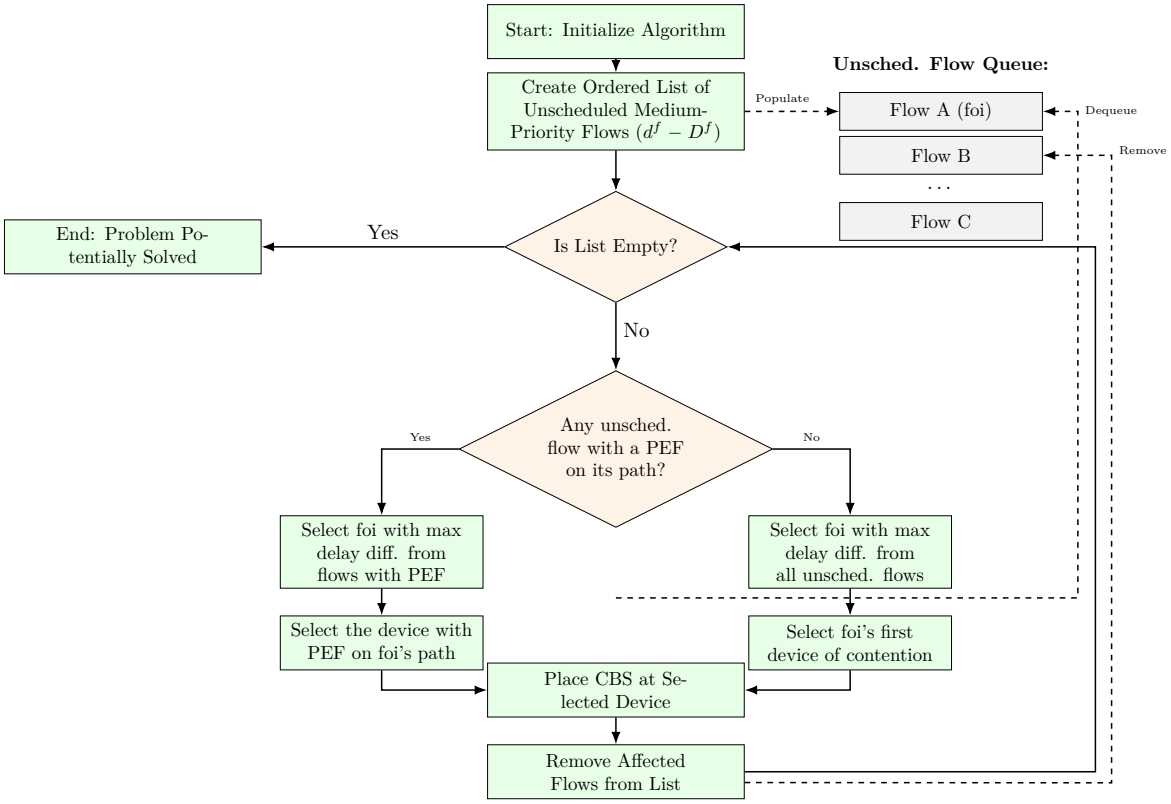


Figure 4.6: FRER-aware device selection strategy for CBS placement

following schedulability condition is derived:

$$\frac{\sum_{f\in\mathcal{F}(op,p)}(b^f + r^f \cdot J^f_{diff})}{I^p_{op}} + \frac{\sum_{k=0}^{p-1} cred^{k,\min}_{op} - L^{>p,\max}_{op}}{\sum_{k=0}^{p-1} I^k_{op} - C} \leq D^p_{op} \cdot \text{margin} \tag{4.14}$$

This expression can be generalized to also account for the case where no PEF is implemented at (op, p) by multiplying the new burstiness component by PEF^p_{op}, which is equal to 1 if a PEF is implemented and 0 otherwise. Thus,

$$\frac{\sum_{f\in\mathcal{F}(op,p)}(b^f + \text{PEF}^p_{op} \cdot r^f \cdot J^f_{diff})}{I^p_{op}} + \frac{\sum_{k=0}^{p-1} cred^{k,\min}_{op} - L^{>p,\max}_{op}}{\sum_{k=0}^{p-1} I^k_{op} - C} \leq D^p_{op} \cdot \text{margin} \tag{4.15}$$

Rearranging Equation 4.15, we derive a local bound for IdleSlope I^p_{op}:

$$I^p_{op} \geq \frac{\sum_{f\in\mathcal{F}(op,p)}(b^f + \text{PEF}^p_{op} \cdot r^f \cdot J^f_{diff})}{D^p_{op} \cdot margin - \frac{\sum_{k=0}^{p-1} cred^{k,\min}_{op} - L^{>p,\max}_{op}}{\sum_{k=0}^{p-1} I^k_{op} - C}} \tag{4.16}$$

Through Equation 4.16, we can compute the minimum IdleSlope for a CBS placed at (op, p), considering the potential burstiness increase due to FRER.

Finally, the margin computation remains the same as in Chapter 3, where it is used to compensate for the simplification of ignoring burst propagation and considering the minimum upper delay bound instead of the upper delay bound classically computed using NC. The margin is initialized to 1 and is iteratively decreased by a constant rate ρ until a feasible solution is found or until it reaches a theoretical minimum, $\max(\underline{d}^p_{op}/D^p_{op})$, where D^p_{op} is the local deadline calculated via Equation 4.4. This minimum represents the tightest possible timing constraint; if the margin falls below this limit, no feasible solution can be found with the current approach.

With these three elements, we may now detail the complete FRER-aware Partial CBS Deployment algorithm.

4.4.4 Adapting the Existing Partial CBS Deployment Algorithm for FRER-Awareness

The FRER-Aware Partial CBS Deployment algorithm, detailed in Algorithm 1, systematically places and configures CBS (CBS) to ensure network schedulability, while clustering these instances to a reduced set of the Switches in the network. It operates on six inputs: the margin variable m, used for calculating available bandwidth; its constant decrease rate ρ, applied during reconfigurations; the sets of currently unscheduled shaped flows (*unschedShaped*) and non-shaped flows (*unschedNonShaped*); the set of available PEF-enabled devices (*unexploredFRER Ops*); the set of devices already considered in previous iterations (*already-*

ExploredDevs); and *foi_prev* which corresponds to the flow of interest from the previous execution of the algorithm.

Lines 2–10 show the reconfiguration process, which is triggered in case the previous CBS placement did not yield a schedulable result for shaped flows. This process is virtually unchanged from the original algorithm, except for the IdleSlope computation in Line 6, which now uses Equation 4.16 to account for FRER's impact on traffic burstiness.

If all shaped flows are schedulable but some non-shaped flows are not, the algorithm proceeds to its main CBS placement phase (Lines 11–38). The goal here is to strategically introduce new CBS instances to regulate the problematic non-shaped flows. Several sets are initialized for this task: `devsToExclude` (Line 11) tracks devices that are unsuitable for new CBS instances; `shapedFlows` (Line 12) records the flows that become shaped during this process; and `FRERops` (Line 13) identifies output ports that use PEF and are traversed by the highest-priority unscheduled flows, making them prime candidates for CBS deployment.

A `while` loop (Line 14) drives the placement process, continuing as long as there are unscheduled non-shaped flows. In each iteration, the algorithm selects a flow of interest, `foi` (Line 16), which is the most critical flow to fix, the flow with the highest difference between its delay and deadline. The selection process prioritizes flows that traverse FRER-enabled ports if such ports are available, as these locations are more likely to benefit from CBS placement due to the burstiness introduced by frame elimination. Thus, if the `FRERops` set is populated, the selection of `foi` is restricted to flows that pass through these ports:

$$foi = \arg \max_{f \in \texttt{unschedNonShaped} | \exists op \in \phi^f \cap \texttt{FRERops}} \left(d^f - D^f \right) \tag{4.17}$$

In case all unscheduled non-shaped flows do not traverse any FRER-enabled ports, the selection falls back to the original criterion:

$$foi = \arg \max_{f \in \texttt{unschedNonShaped}} \left(d^f - D^f \right) \tag{4.18}$$

The condition in Lines 18–20 is there to identify if CBS placement at ports with PEF implementation made it possible to reduce the number of unshaped and unscheduled flows, thus advancing towards the solution. If CBS placement (or failure to do so) did not go in that sense, exploration is stopped to externally reset the network of CBS placement and trigger the Busrtiness-Aware Partial CBS Deployment Algorithm instead. This way, we ensure that at worse, the algorithm fallbacks into the original framework. As in Algorithm 1, if the foi keeps being the same as in the previous execution of the algorithm and all of its path has been explored, the algorithm concludes that no solution can be found and returns null (Lines 20–22).

Once the `foi` is identified, the algorithm determines the best location to place a CBS (Line 24). The selection depends on whether *foi* traverses a FRER-enabled port. If this is the case, then the selected device will be the first device implementing a PEF where *foi* encounters higher priority unshaped traffic. If not, then it will simply be the first time *foi*

encounters higher priority traffic along its path. The output ports on this candidate device, `dev`, are then sorted into a list `ops` based on the ascending order of the difference between delay and deadline of the flows that traverse them, prioritizing the ports with PEF (Line 25).

The algorithm then iterates through each output port in `ops` (Lines 27–34). Following the exact logic from Algorithm 1, where the main difference lies in Line 29, which now uses Equation 4.16 to compute the IdleSlope for the new CBS instance, considering FRER's effects. The remaining steps reproduce the logic from the original algorithm, where CBS instances are placed, `shapedFlows` is updated, and `unschedNonShaped` is re-evaluated depending on the placed CBS influence on the flows present in this list.

The main differences with the FRER-blind Algorithm described in Chapter 3 are written in bold when detailing the Algorithm 2.

The FRER-aware algorithm was designed as a superset of the strategy presented in the previous chapter. In scenarios where FRER is disabled or not required, the specific logic targeting elimination points is naturally bypassed, and the heuristic reverts to the behavior of the original Burstiness-aware algorithm. This backward compatibility ensures that the framework provides a unified solution for partial CBS deployment, applicable to both standard and redundant TSN networks without requiring distinct frameworks.

4.4.4.1 Example use of the algorithm

To illustrate the proposed algorithm, consider the network of Figure 4.4, traversed by flows from Table 4.2.

1. Aside from initializations, since this is the first execution of the algorithm, the first operation done is the selection of the *foi*. In this case, it corresponds to f_2, as it is the flow with the highest deadline slack under NP-SP. The deadline slack for f_2 is $13\mu s$ while f_1 has a deadline slack of $10\mu s$.

2. Then the device selection is done. Since f_2 traverses a PEF at $SW1_2$, the selected device is $SW1_2$, as it is the first device where f_2 encounters higher priority traffic, which is f_0.

3. Amongst the ports of $SW1$, the only port being traversed by unscheduled flows is $op = SW1_2$, Thus a CBS is set to be placed on $p = 0$ as it is the highest non-shaped priorirty being traversed by schedulable flows.

4. As for the first iteration, $margin = 1$, the minimum IdleSlope is then computed using Equation 4.16, which returns $I^0_{SW1_2} = 586$ Mbps, which respects the upper bound of 750 Mbps.

5. Since all the unscheduled flows were affected by this CBS placement, *Untreated unscheduled flows*: $\emptyset$, the algorithm terminates and the solution is verified.

Algorithm 2 FRER-Aware Partial CBS Algorithm

```
1: function FRERAwarePartialCBSDeployment(m,ρ,unschedShaped, unschedNonShaped, unexplored-
   FREROps, alreadyExploredDevs, foi_prev)
2:     if unschedShaped ≠ ∅ then
3:         // Perform Reconfiguration
4:         m ← m − ρ
5:         if m ≤ m_min then
6:             return null                                   ▷ No solution could be found
7:         end if
8:         recomputeIdleSlopes(m)
9:         return network configuration
10:    end if
11:    devsToExclude ← alreadyExploredDevs
12:    shapedFlows ← ∅
13:    FRERops ← get_FRER_ops(UnexploredFREROps)
14:    while unschedNonShaped ≠ ∅ do
15:        // Place CBS
16:        foi ← FRERAwareMostConstrainedFlow(unschedNonShaped, FRERops)
17:        if foi == foi_prev && (φ^f \ {src, dst}) ∈ devsToExclude then
18:            if ∀op ∈ φ^f, PEF^p_op == 1 then
19:                return stop_flag
20:            else
21:                return null                               ▷ No solution could be found
22:            end if
23:        end if
24:        dev ← FRERAwareGetDev(foi, devsToExclude, FRERops)
25:        ops ← getOrderedOps(dev)
26:        // Configure CBS
27:        for all op ∈ ops do
28:            p ← highestNonShapedPriority(op)
29:            I^p_op ← FRERAwareComputeIdleSlope(op)
30:            if Σ_∀p I^p_op ≤ 0.75 · C then
31:                CBS^p_op ← 1
32:                shapedFlows.add({f | f ∈ F(op, p)})
33:            end if
34:        end for
35:        DA ← getDirectlyAffectedFlows()
36:        IA ← getIndirectlyAffectedFlows()
37:        unschedNonShaped.remove(DA ∪ IA)
38:    end while
39:    return network configuration
40: end function
```

Flow	Deadline (ms)	Computed Delay Bound (ms)
f_0	5	1.65
f_1	2	1.19
f_2	3	2.22
f_3	-	8.97

Table 4.5: Delay bounds for the example network under FRER-aware Partial CBS Deployment algorithm.

This solution returns a configuration that renders all flows schedulable while minimizing the number of TSN devices and accounting for the burst increase caused by FRER. Table 4.5 shows the delay results for this example network after applying the FRER-aware Partial CBS Deployment algorithm, confirming that all flows meet their deadlines.

4.4.5 Complexity Analysis Computation

The FRER-aware Partial CBS Deployment algorithm, as detailed in Algorithm 2, exhibits a computational complexity identical to the one presented in Theorem 3.1.

Theorem 4.1
The computational complexity of the FRER-Aware Partial CBS Deployment algorithm is:

$$O\left(|\mathcal{F}|^3 \cdot |DEV|^2 + |\mathcal{F}|^2 \cdot |DEV|^2 \cdot \log(|DEV|)\right)$$

where $|\mathcal{F}|$ is the number of flows and $|DEV|$ is the number of devices in the network.

Proof. The proof stems from the proof of Theorem 3.1, as Algorithm 2 shares the same structure of Algorithm 1. Thus, this proof will focus on the changed functions that are in bold in Algorithm 2.

- **Initial Reconfiguration:** The complexity of the reconfiguration phase (Lines 2–10) is primarily influenced by the recomputation of IdleSlopes (Line 8). In the FRER-aware version of the algorithm, this operation (See Equation 4.16) still involves iterating over at most on iterating every flow to recalculate the IdleSlope values based on the updated margin. Given this is explored through every device and output port, the complexity remains:

$$O(|DEV|^2 \cdot |\mathcal{F}|) \tag{4.19}$$

- **FRER-aware Most Constrained Flow Selection:** The selection of the most constrained flow (Line 16) now includes a check to see if the flow traverses any FRER-enabled ports. This additional check involves iterating through the list of FRER-enabled ports for each unscheduled non-shaped flow. Although this check does not change the fact that at most, all flows will be checked, as in the worst case, all flows either are FRER-enabled or not. Thus, the complexity remains:

$$O(|\mathcal{F}|) \tag{4.20}$$

- **FRER-aware Device Selection:** The device selection process (Line 24) now incorporates a check for FRER-enabled ports along the flow's path. This involves iterating through the devices in the flow's path and checking against the list of FRER-enabled ports. However, since this operation still involves iterating through the devices in the

flow's path, the overall complexity remains:

$$O(|DEV|) \tag{4.21}$$

- **FRER-aware IdleSlope Computation:** The computation of the IdleSlope (Line 29) now uses Equation 4.16, which includes an additional term for burstiness due to FRER. However, this computation still involves iterating over the flows that traverse the output port to sum their burstiness and rates. Thus, the complexity remains:

$$O(|\mathcal{F}|) \tag{4.22}$$

Since the structure of the algorithm remains unchanged, and the added operations do not increase the order of complexity, the overall computational complexity of the FRER-aware Partial CBS Deployment algorithm remains:

$$O\left(|\mathcal{F}|^3 \cdot |DEV|^2 + |\mathcal{F}|^2 \cdot |DEV| \cdot \log(|DEV|)\right)$$

□

4.5 Validation over an Automotive Use-Case

To validate the effectiveness of the FRER-aware Partial CBS Deployment framework, we apply it to the same automotive network topology used in Chapter 3. However, the traffic characteristics have been adapted to include redundant streams, creating a scenario where FRER-induced burstiness is a significant factor. The main objective is to demonstrate that by explicitly accounting for FRER, specifically the burstiness generated at elimination points, we can achieve network schedulability with fewer resources and better delay performance than a FRER-blind approach.

4.5.1 Network Topology and Traffic Model

The evaluation uses the topology presented in Figure 3.7. The traffic profile, detailed in Table 4.6, is designed to create a challenging contention scenario. High-priority AVB flows ($p = 0$) are replicated using FRER at the switch after their source, utilizing disjoint paths and merging at the switch prior to their destination. These flows compete with latency-critical CAN traffic ($p = 1$) and bandwidth-intensive Best-Effort traffic ($p = 2$). The AVB flows are highly bursty, and this burstiness is exacerbated by the Frame Elimination function at the merging points, posing a risk to the deadline satisfaction of the unshaped CAN flows.

Table 4.6: Network characteristics for the industrial use case with FRER

Flow	Src	Dst	Repl.?	Frame size (B)	Frames/period	Period (ms)	D^f (ms)	Prio
AVB_1	src_av_1	dst_av_{1-2}	Y	1390	2	1.25	1.25	0
AVB_2_0	src_av_{3-7}	HPC_Host, HPC_Host_AD	Y	1038	15	33	100	0
AVB_2_1	src_av_{4-7}	HPC_Host	Y	1038	15	33	100	0
AVB_3	src_av_8	HPC_Host, HPC_Host_AD	Y	1438	30	33	33	0
CAN_1_0	ZCP1_Host	ZCP{2–5}, HPC_Host, HPC_Host_AD	N	46	1	1	1	1
CAN_1_1	HPC_Host	ZCP{2–5}, HPC_Host_AD	N	46	1	1.5	1.5	1
CAN_2_0	ZCP1_Host	ZCP{2–5}, HPC_Host, HPC_Host_AD	N	78	1	1	1	1
CAN_2_1	HPC_Host	ZCP{1–5}, HPC_Host_AD	N	78	1	1.5	1.5	1
CAN_3_0	ZCP1_Host	ZCP{2–5}, HPC_Host, HPC_Host_AD	N	86	1	1	1	1
CAN_3_1	HPC_Host	ZCP{1–5}, HPC_Host_AD	N	86	1	0.6	0.6	1
CAN_4_0	ZCP1_Host	ZCP{2–5}, HPC_Host, HPC_Host_AD	N	42	1	1.1	1.1	1
CAN_4_1	HPC_Host	ZCP{1-5}, HPC_Host_AD	N	42	1	0.8	0.8	1
FILE	HPC_Host	Tool	N	1500	85	12	200	2

4.5.2 Results and Analysis

We compare the performance of our **FRER-Aware Partial CBS Deployment** against three baselines:

1. **FRER-Blind Partial CBS Deployment:** The burstiness-aware strategy from Chapter 3, which places CBS based solely on path interference without knowledge of PEF locations.
2. **Full CBS Deployment:** A configuration where every switch is TSN-capable and CBS is enabled on all ports traversed by high-priority traffic.
3. **No CBS (NP-SP):** A legacy baseline using only strict priority scheduling.

Costs

Table 4.7 details the deployment locations and configuration of CBS instances for both partial strategies.

The results highlight the cost efficiency of the FRER-aware approach. While the FRER-blind strategy requires deploying CBS on 3 distinct devices (HPC, HPC_AD, and ZCP1), the FRER-aware strategy successfully clusters the solution onto only 2 devices (HPC and HPC_AD). In terms of hardware capabilities, this represents a significant reduction in network cost; compared to a Full CBS deployment (7 switches), the FRER-aware approach reduces the number of required TSN-enabled switches by approximately 71%. Furthermore, by targeting the specific source of burstiness, the elimination nodes, the FRER-aware algorithm instantiates fewer CBS instances (5 vs. 6) compared to the FRER-blind one, simplifying the overall network configuration overhead.

Table 4.7: Deployed CBS Parameters under FRER-Blind and FRER-Aware heuristics

Output port	**FRER-Blind Partial CBS**	**FRER-Aware Partial CBS**
HPC to HPC_Host	151 Mbps	151 Mbps
HPC to ZCP1	170 Mbps	170 Mbps
HPC to ZCP2	144 Mbps	144 Mbps
HPC to ZCP3	58 Mbps	58 Mbps
HPC_AD to HPC_AD_Host	170 Mbps	170 Mbps
ZCP1 to HPC	346 Mbps	—

Performance and Schedulability

We evaluate the impact of these configurations on traffic latencies using Cumulative Distribution Functions (CDFs) of the computed delay bounds.

Impact on Shaped Traffic ($p = 0$): Figure 4.7 illustrates the delay bounds for the high-priority AVB flows. A distinct hierarchy in performance is observable. The Full CBS deployment exhibits the highest worst-case delays; this confirms the *blocking effect* discussed in Chapter 3, where a sequence of multiple CBS accumulates delay. The FRER-aware strategy achieves the best performance, reducing the maximum delay bounds by up to 84% compared to Full CBS and approximately 7% compared to the FRER-blind strategy. This improvement stems from the placement strategy: by shaping traffic immediately at the PEF (the point of burstiness injection), the FRER-aware approach prevents the burst from propagating and accumulating latency downstream, allowing for more relaxed shaping parameters elsewhere.

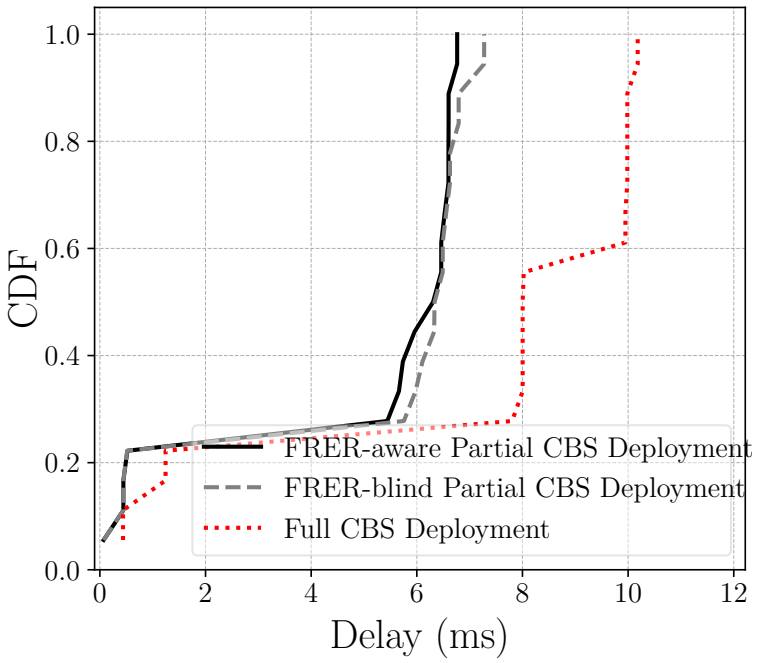


Figure 4.7: Shaped Flows ($p = 0$): Comparison of deployment strategies.

Impact on Unshaped Traffic ($p = 1$): Figure 4.8 analyzes the critical CAN flows. Under the No CBS (NP-SP) baseline, these flows fail to meet their deadlines due to contention caused by the bursty AVB traffic. Both Partial CBS strategies successfully render the network schedulable. It is interesting to observe that the FRER-blind approach yields slightly lower delays for $p = 1$ flows than the FRER-aware approach. This is an expected effect of the *over-deployment*, as more CBS instances improve unshaped flows delays. Consequently, because the FRER-blind approach deploys an additional shaper (ZCP1 to HPC), it slows down the high-priority $p = 0$ traffic more aggressively. However, the FRER-aware approach achieves the main objective of guaranteeing that all $p = 1$ flows meet their hard deadlines, while using fewer CBS and providing better delay bounds to the high-priority flows.

In summary, the validation confirms that the FRER-aware heuristic effectively solves the

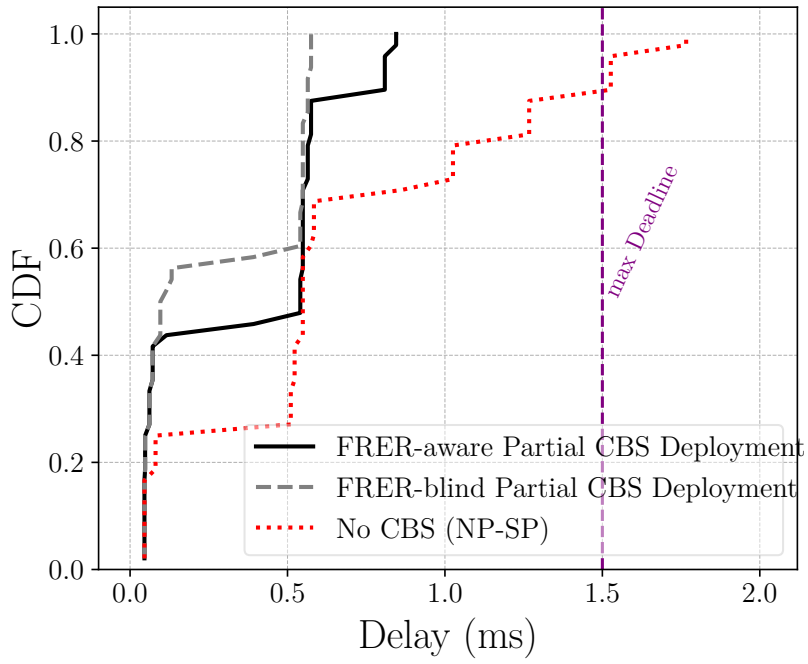


Figure 4.8: Unshaped Flows ($p = 1$): Comparison of deployment strategies.

specific burstiness challenges introduced by frame elimination. In addition, it leads to a good performance trade-off in a FRER-enabled network, allowing effective Partial CBS deployment that minimizes costs while ensuring all traffic schedulability.

4.6 Conclusions

In this chapter, we have addressed the specific challenges of deploying CBS in TSN networks that implement FRER. We have established that while FRER enhances reliability, the elimination function introduces significant burstiness increase that compromises the schedulability of time-critical flows. To counter this, we have proposed a FRER-aware Partial CBS Deployment framework that prioritizes the deployment of CBS at elimination points and adapts IdleSlope configuration to account for the burstiness increase due to FRER.

Through the validation on an automotive use-case, we have shown that this FRER-aware strategy is more cost-effective than burstiness-aware,i.e., FRER-blind, approach, achieving network-wide schedulability with fewer TSN-enabled devices and reducing worst-case delay bounds for high-priority traffic.

Despite these promising results, it is important to acknowledge the lack of generalization regarding the validation of the algorithm's efficiency. While the heuristic provides a feasible solution that outperforms standard baselines, we currently lack a quantitative assessment

of how close this solution is to the global optimum. Therefore, the optimality gap of our proposed configuration needs to be quantified.

Furthermore, the scalability of the proposed approach has to be fully evaluated. While the framework proved effective for the analyzed automotive topology, we need to analyze how the heuristic scales with significantly larger network sizes and higher flow densities.

In the next chapter, we tackle these challenges using synthetic use cases to better assess the optimality and scalability of our proposed CBS deployment framework.

Chapter 5

Improved Partial CBS Deployment Framework and Validation

Contents

5.1 Tools for Optimality and Scalability Evaluation 90
5.1.1 Brute Force Solver for Optimality assessment 90
5.1.2 Synthetic Network Generation 91
5.2 Burstiness-Aware Partial CBS Deployment Assessment 93
5.2.1 Optimality Evaluation 93
5.2.2 Improved Burstiness-Aware Partial CBS Deployment Framework 97
5.2.3 Schedulability Evaluation 101
5.2.4 Scalability Evaluation 101
5.2.5 Re-evaluation of the Industrial Use-case 104
5.3 Assessment of the FRER-Aware Partial CBS Deployment Framework 105
5.3.1 Optimality Evaluation 106
5.3.2 Improved FRER-aware Partial CBS Deployment Framework 108
5.3.3 Schedulability Evaluation 112
5.3.4 Scalability Evaluation 113
5.3.5 Re-evaluation of the Industrial Use-case 114
5.4 Conclusions 115

In the preceding chapters, we established a cost-efficient approach for deploying the CBS in TSN. Chapter 3 introduced the foundational *Partial CBS Deployment* strategy, designed to mitigate deployment costs while preserving the benefits of traffic shaping. Chapter 4 expanded this methodology to be *FRER-aware*, explicitly addressing the burstiness increase due to FRER. While we demonstrated that our approach ensures schedulability with a reduced number of TSN devices in an industrial use-case, further evaluation needs to be conducted to assess the optimality and the scalability of our proposal.

This chapter addresses these open questions by rigorously validating the framework against exact methods and testing its performance on large-scale networks.

To achieve this, we introduce a *Brute Force* solver to determine the optimal baseline solution and a synthetic network generator to produce a diverse range of test scenarios. Based

on the insights gained from this deeper evaluation, we have identified some limitations in our initial heuristic and proposed the *Improved Partial CBS Deployment Framework*. This enhanced framework is then subjected to the same validation bench to demonstrate its superior performance in terms of both solution quality and computational scalability.

The chapter is organized as follows:

- Section 5.1 details the evaluation tools, including the guided Brute Force solver used to establish an optimal baseline and the synthetic network generator designed to produce diverse topologies and traffic profiles.
- Section 5.2 presents a comprehensive assessment of the *Burstiness-Aware Partial CBS Deployment* framework. We first evaluate the heuristic to quantify the optimality gap, then we introduce the *Improved Partial CBS Deployment Framework* to address identified issues, and finally validate the improved approach in terms of solution quality and computational scalability.
- Section 5.3 assesses the *FRER-Aware Partial CBS Deployment Framework*, expanding the evaluation to FRER-enabled networks.
- Section 5.4 summarizes the validation results and concludes the chapter.

5.1 Tools for Optimality and Scalability Evaluation

To rigorously validate our heuristic, robust evaluation tools are required. Ideally, the optimization problem defined in Chapter 3 would be solved using exact analytical methods. However, as discussed in Sections 3.3.2 and 4.3.1, the considered large-scale problems render traditional ILP solvers inefficient. Consequently, we employ a tailored brute force approach to establish a baseline to assess the optimality of our framework. Additionally, to validate our framework under diverse conditions, we utilized a synthetic network generator to create a wide array of topologies and traffic patterns.

5.1.1 Brute Force Solver for Optimality assessment

To benchmark our heuristic against a known global optimum, we have implemented a guided brute force solver. Instead of an exhaustive, blind search of the entire solutions space, our method iterates on the *target cost* (number of TSN devices). This way we can identify the minimal number of devices required to achieve schedulability more efficiently than a classic brute force method.

To do so, we consider an extra constraint for the problem set in Section 3.3. The constraint involves fixing the number of allowed TSN devices, K:

$$K - 1 < \sum_{dev \in DEV} TSN_{dev} \leq K, \quad K \in \{1, \ldots, |DEV|\} \tag{5.1}$$

The algorithm is initialized with $K = 1$ and it exhaustively checks if any configuration with exactly one TSN device satisfies the schedulability constraints. If no valid configuration is found, K is incremented. This iterative expansion ensures that the first feasible solution found is guaranteed to be optimal in terms of device count. Figure 5.1 illustrates this process. For a given K, the solver generates a candidate combination of devices for which all combinations of CBS instances are tested, before exploring a new combination of devices.

The current configuration to test is passed to the *Schedulability Analysis* module (using FP-TFA++, see Section A.6.4). If the configuration is feasible, the process terminates, returning the optimal solution. If not, the solver continues until a solution is found or the search space for K is exhausted.

IdleSlopes are computed based on Note 3.5 of Chapter 3, applied only to the active CBS instances.

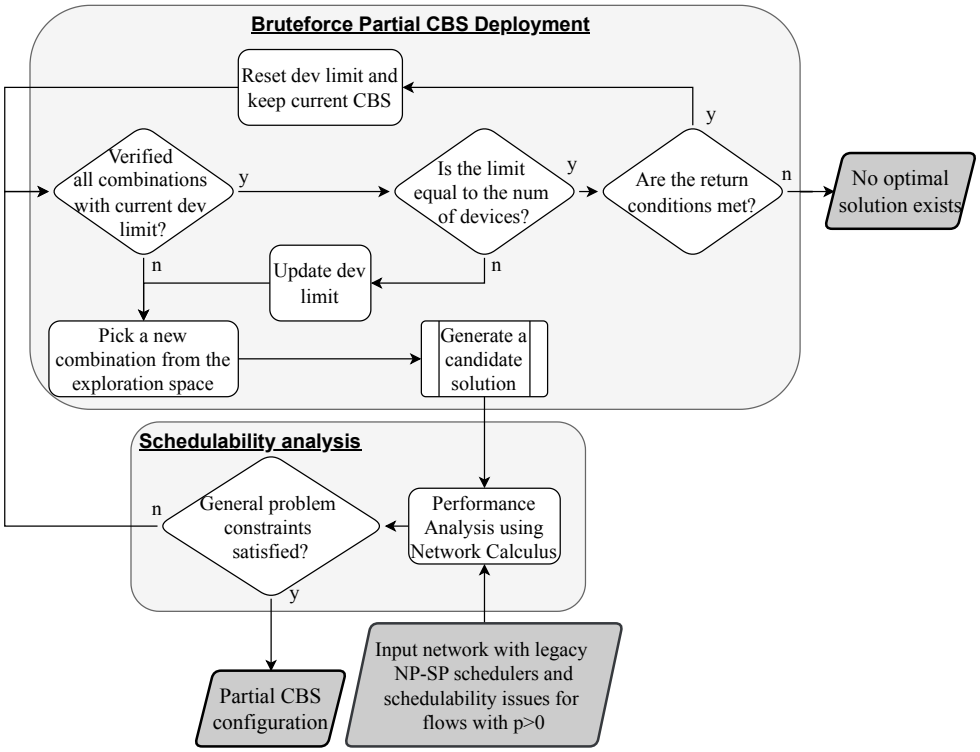
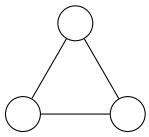


Figure 5.1: Brute force Comparison Methodology

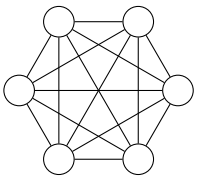
5.1.2 Synthetic Network Generation

To validate the framework under general conditions, we have developed a synthetic benchmark generator capable of producing diverse network configurations varying in topological scale and traffic load. We selected Mesh topology (Figure 5.2) that offers multiple paths to enable FRER

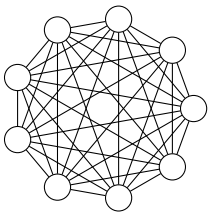
within representative industrial networks [49], [52].



(a) Small Mesh (3 SW)



(b) Medium Mesh (6 SW)



(c) Large Mesh (9 SW)

Figure 5.2: The topologies used in the experiments, shown in the three different used sizes.

We defined three network sizes (Small, Medium, Large) as detailed in Table 5.1. Traffic load was defined by the number of flows and their load, categorized into Low ($< 20\%$), Medium (40%–60%), and High ($> 60\%$) regimes. Traffic is classified into three priority levels (Table 5.2), designed to create realistic contention: High priority traffic is bursty, creating interference for the non-bursty Medium and Low priority classes, thereby necessitating CBS deployment. Routes were assigned via shortest-path between randomly generated source and destination.

Table 5.1: Network Size Configurations

Configuration	**Switches**	**End Systems**
Small-sized	3	3
Medium-sized	6	6
Large-sized	9	9

Table 5.2: Traffic Profile Characteristics

Priority	**Size (Bytes)**	**Period (ms)**	**Burst (Frames)**
High (p=0)	750–1000	1.2–1.5	5–15
Medium (p=1)	150	1.2	1
Low (p=2)	1500	12	60

The evaluation is structured around two primary scenarios:

1. **Fixed Size, Variable Load:** Uses the Medium-size topology with varying load to assess performance degradation under congestion.

2. **Fixed Load, Variable Size:** Maintains Medium load while scaling the topology to assess algorithmic scalability.

For each configuration, 100 network instances were generated. We specifically selected instances where high-priority flows were initially schedulable, to focus on schedulability problem of lower-priority traffic that needs to be solved using CBS.

5.2 Assessment of the Burstiness-Aware Partial CBS Deployment Framework

We first evaluate the *Burstiness-aware Partial CBS Deployment* heuristic (presented in Chapter 3) against the Brute Force baseline. This comparison focuses on non-FRER scenarios to isolate the performance of the core placement logic.

5.2.1 Optimality Evaluation

Figure 5.3 and Figure 5.4 present the distribution of the evaluated networks as a function of the number of TSN devices that guarantees schedulability constraints, under our heuristic and Brute force method when varying network sizes and loads, respectively.

While the framework consistently outperforms a Full CBS deployment, often requiring less than half the devices, a comparison with the optimal Brute Force solution reveals some limitations. As network size increases in Figure 5.3, the number of deployed TSN devices under our heuristic move away from the global optimum solution and the number of unsolved cases is around 20%.

The impact of network load is even more pronounced (Figure 5.4). In high-load scenarios, the heuristic set more than double the optimal number of TSN devices, converging towards a Full Deployment solution. Counter-intuitively, we observe a higher failure rate in Medium load scenarios than in High load ones. This anomaly suggests that specific, complex contention patterns at medium load may be particularly challenging for the current heuristic logic, leading us to conclude that this issue is linked to the network generation process rather than the heuristic's behavior.

To further quantify the divergence between the solutions found under the brute force approach and our heuristic, we define the *optimality gap*:

$$\text{Optimality Gap} = \frac{TSN_{dev} \text{ by our framework} - TSN_{dev} \text{ by brute force}}{TSN_{dev} \text{ by Full CBS} - TSN_{dev} \text{ by brute force}} \times 100\% \qquad (5.2)$$

The optimality gap is null when the heuristic converges to the global optimum solution, and equal to 100% when it converges to the full deployment solution.

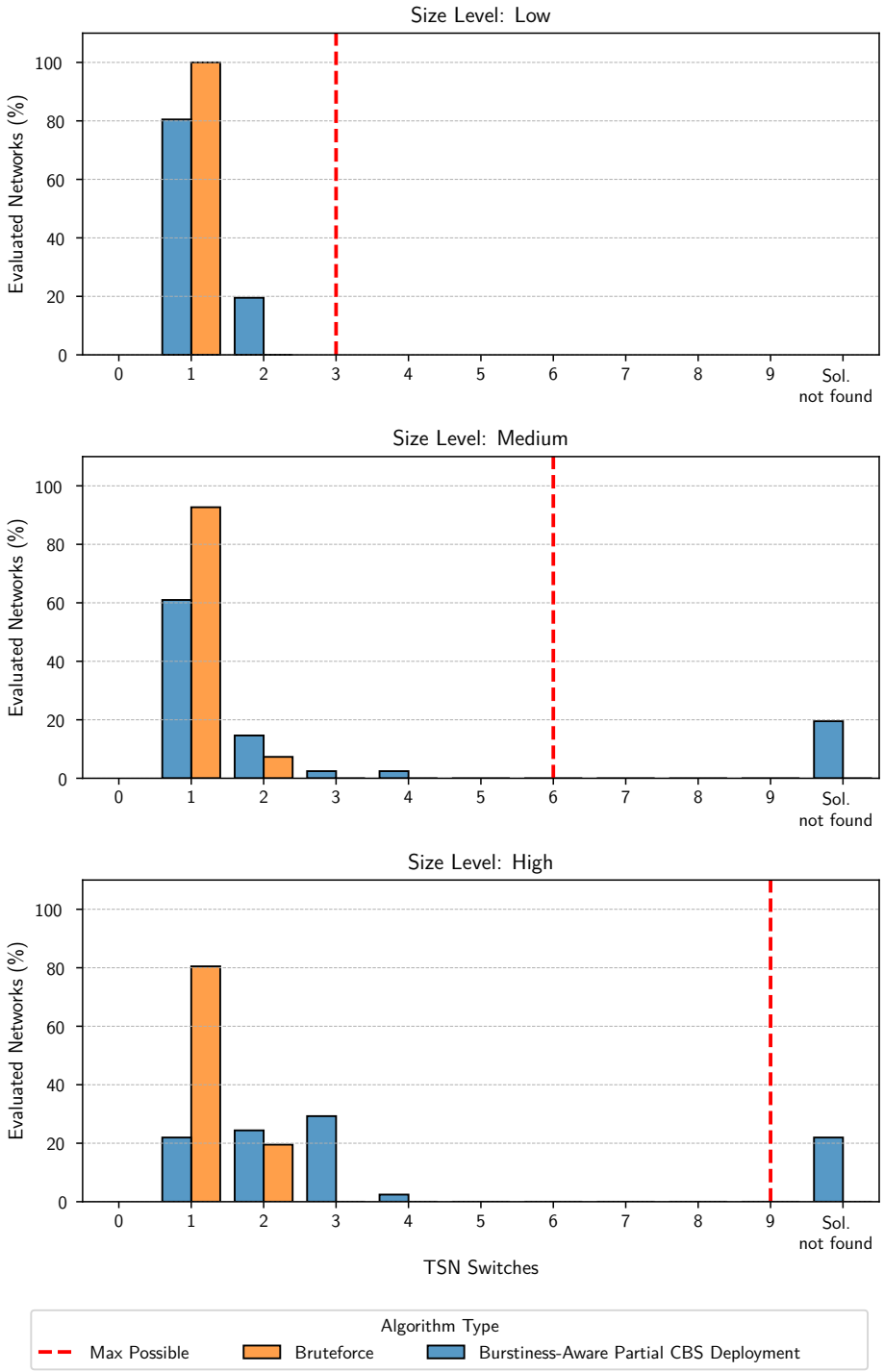


Figure 5.3: Optimality: Heuristic vs. Brute Force (Var. Size)

Figure 5.5 shows the optimality gap across different network loads under a fixed size condition. The results show that as the network size increases, the optimality gap widens. Although the heuristic is still able to produce near-optimal solutions across all sizes, the

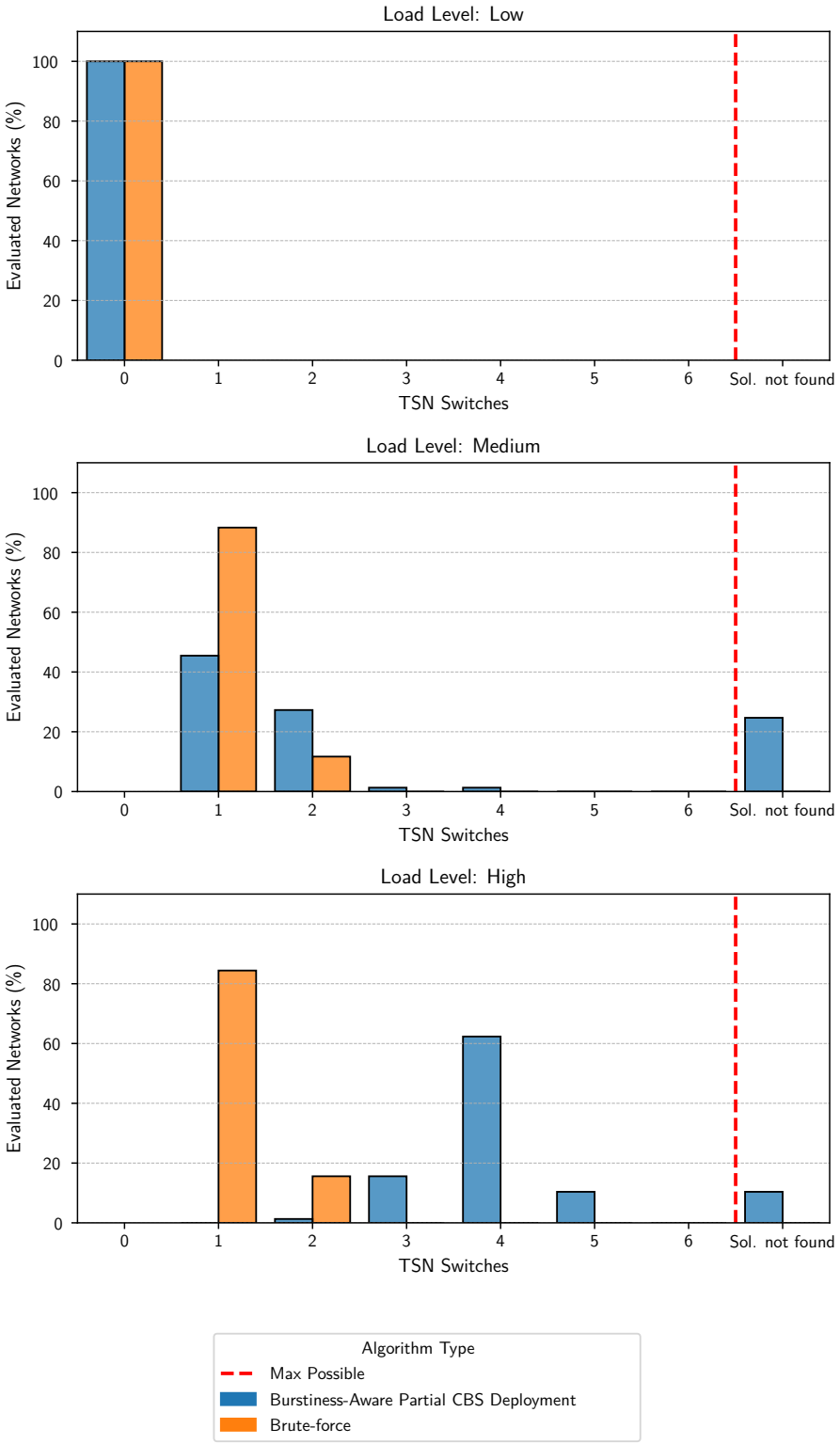


Figure 5.4: Optimality: Heuristic vs. Brute Force (Var. Load)

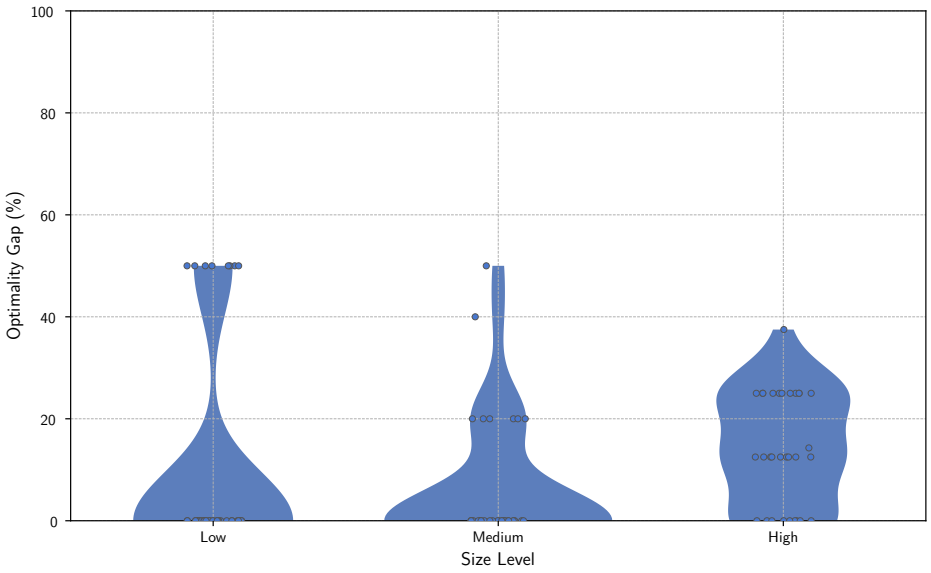


Figure 5.5: Optimality Gap across different network sizes under a fixed load condition.

quantity of these solutions decreases with size, indicating that there is a heuristic performance degradation for large-scale networks.

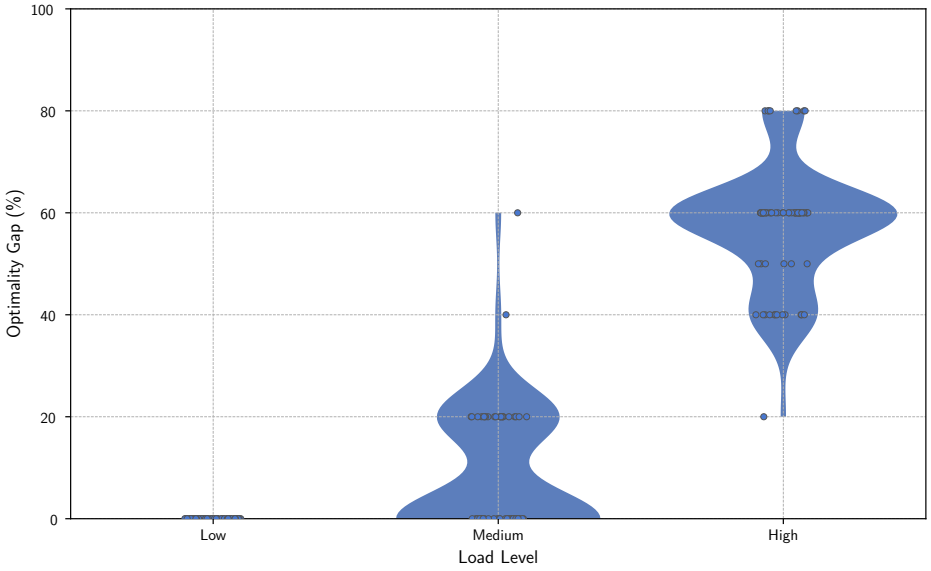


Figure 5.6: Optimality Gap across different network loads under a fixed size condition.

Figure 5.6 illustrates the optimality gap across different network sizes under a fixed load condition. The results indicate that as the network load increases, the optimality gap widens significantly. In contrast to size changes, as load increases, the heuristic cannot maintain near-optimal solutions. This suggests that the heuristic performance is limited in high-load scenarios.

This optimality evaluation reveals two primary limitations of our burstiness-aware heuristic:

- **Premature Search Space Reduction:** In larger networks, making a locally optimal decision (first contention point) often precludes globally optimal placements.
- **Inadequate Contention Handling:** High loads introduce complex interference that a simple topological *first encounter* logic fails to capture effectively.

In the next section, we handle such limitations and propose an improved version of our heuristic.

5.2.2 Improved Burstiness-Aware Partial CBS Deployment Framework

To address the limitations identified in the optimality analysis, we propose a refined heuristic strategy and we re-evaluate its optimality against brute force to validate this improvement.

The optimality evaluation revealed that the main flaw in the original algorithm (Algorithm 1) lies in the selection of the candidate device. Originally, the algorithm selected the *first* device along the path of the Flow of Interest (*foi*) where it encountered higher-priority unshaped traffic. This approach assumes that mitigating burstiness at the earliest opportunity is always optimal. However, the results showed that this is not the case for complex scenarios. The earliest contention point is not necessarily the most critical one.

Therefore, we propose a strategic shift: instead of the *first* encounter, we target the *highest contention* encounter. Specifically, we select the device where the *foi* faces the highest contention of higher-priority unshaped traffic, i.e., the bottleneck in terms of higher-priority rate. By targeting the point of maximum contention, we effectively reduce the most significant source of burstiness. Notably, this strategy preserves the upstream search space, leaving more options open for subsequent iterations if further burstiness reduction is required.

This modification changes the behavior of Line 20 in Algorithm 1. We refer to this as the *Improved Burstiness-Aware Partial CBS Deployment Framework*.

Afterwards, we have re-evaluated the improved heuristic optimality under the same scenarios as the original one.

Figures 5.7 and 5.8 show the distribution of network scenarios in terms of TSN devices number when varying the network size and load, respectively.

The obtained results demonstrate a notable improvement in terms of solution optimality under the refined heuristic with reference to the brute force. The number of required TSN devices drops significantly across all network sizes and loads, consistently hugging the optimal baseline provided by the Brute Force solver. In high-load scenarios, where the original framework struggled most, the improvement is particularly striking.

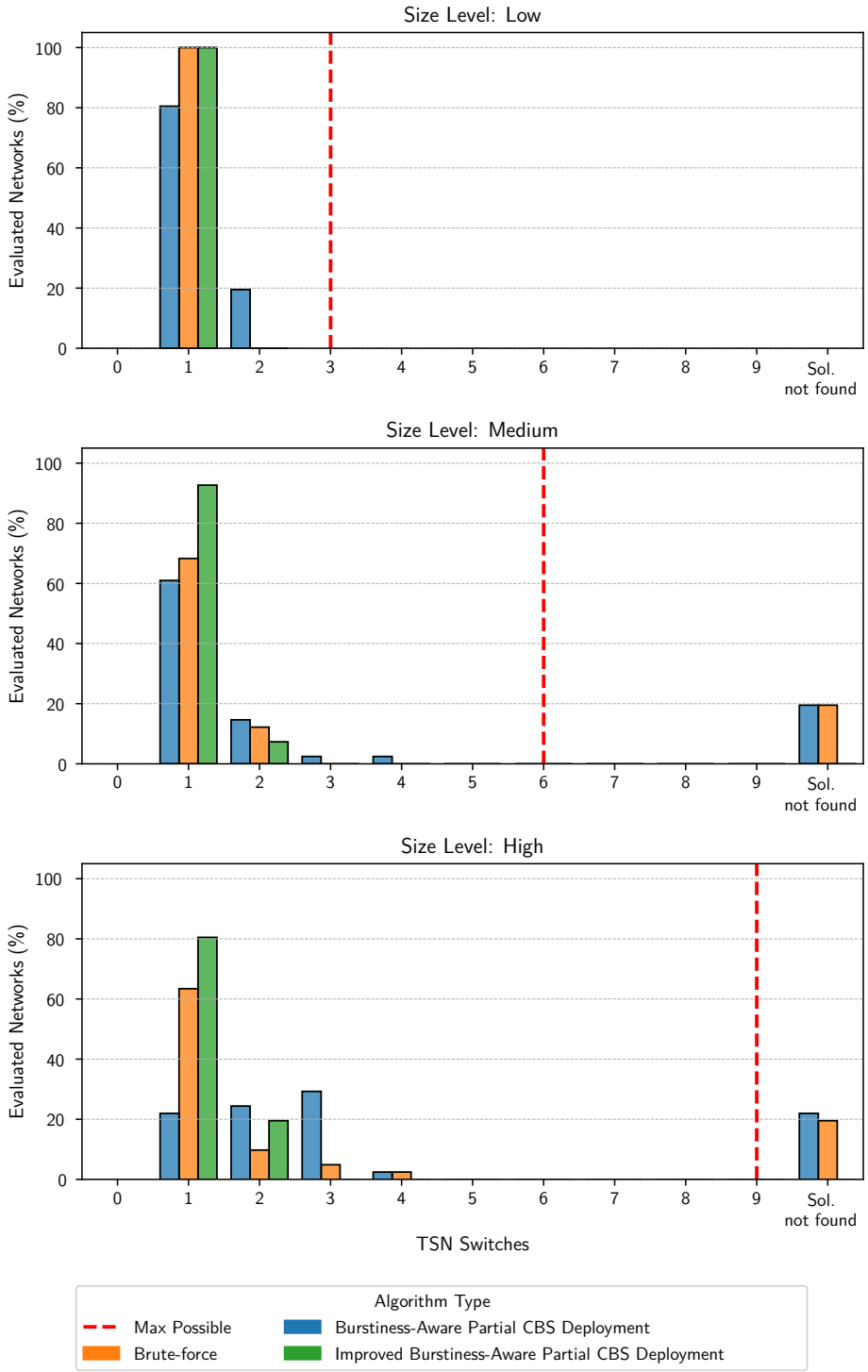


Figure 5.7: Optimality: Improved Heuristic vs. others (Var. Size)

In Figures 5.9 and 5.10, we illustrate the optimality gap of the improved heuristic in reference to the original one, when varying the network size and load, respectively.

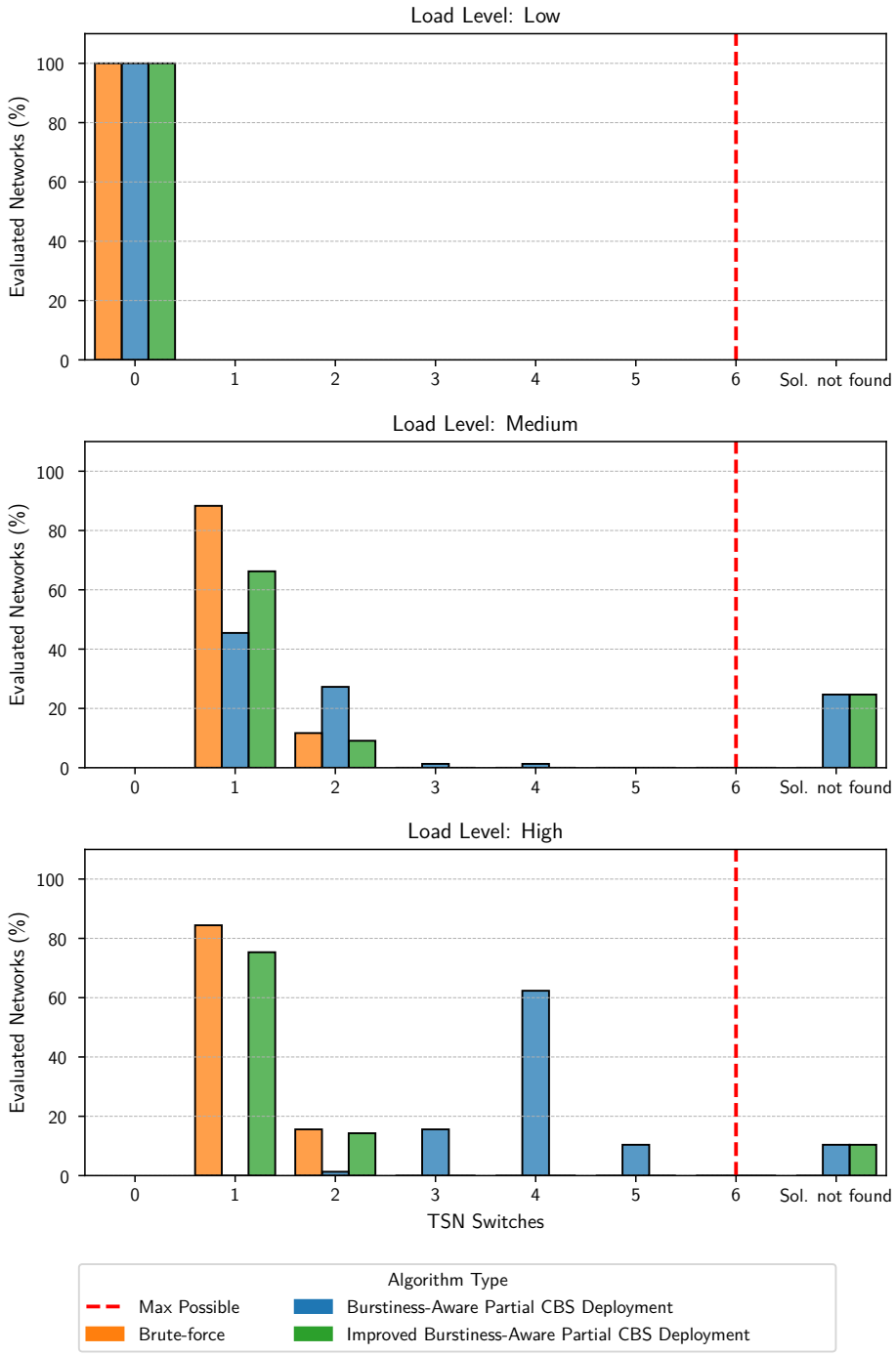


Figure 5.8: Optimality: Improved Heuristic vs. others (Var. Load)

As it can be seen, the improvement of the optimality gap is substantial. Although the average gap was already low for size variations in the original heuristic, the improved heuristic tightens this further, achieving gaps on average of 0%, 2% and 5% for small, medium and large sizes, respectively. Moreover, the improved heuristic effectively eliminates the high optimality

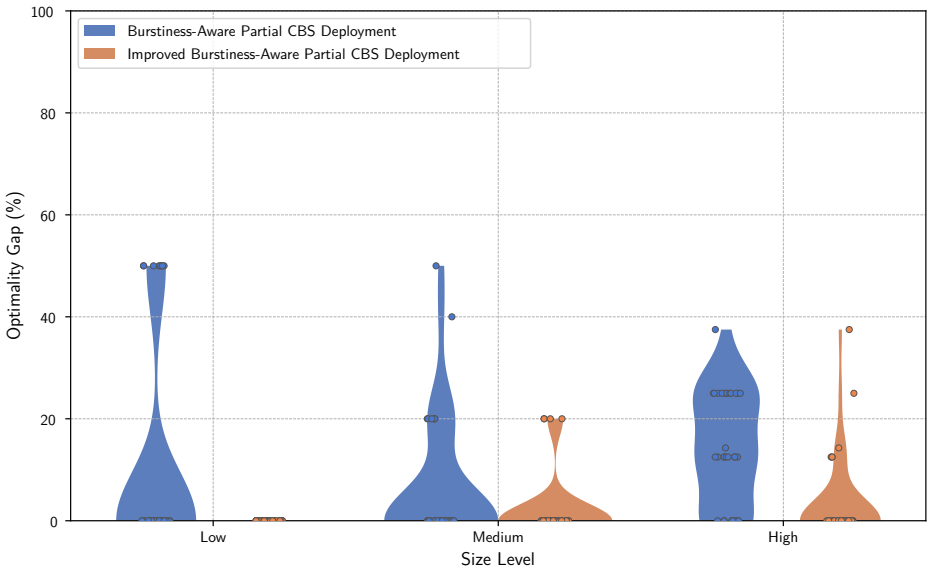


Figure 5.9: Improved Optimality Gap (Variable Load)

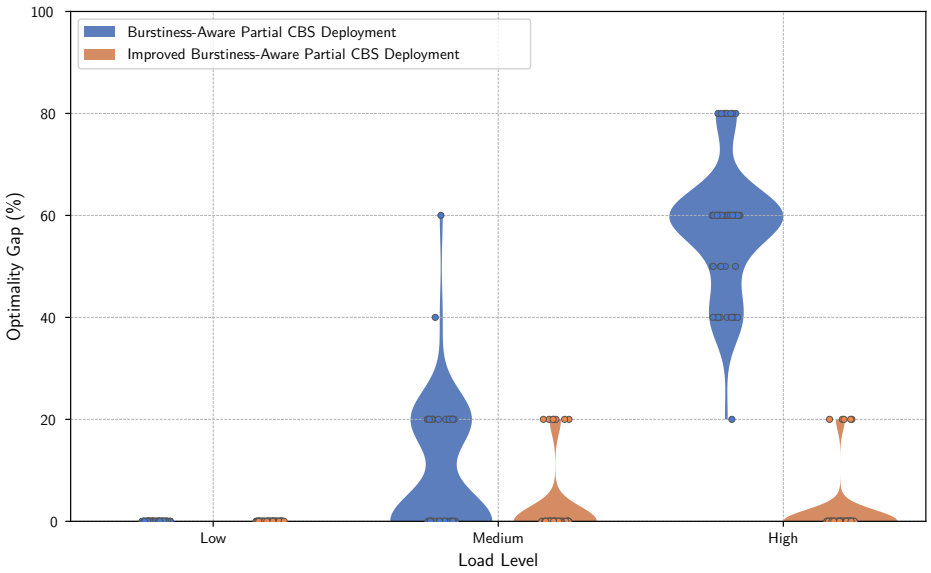


Figure 5.10: Improved Optimality Gap (Variable Size)

gaps observed in load variations. The average gap now stands at 0%, 2% and 1% for low, medium and high load, respectively. These results highlight that the improved placement logic effectively manages contention even in congested scenarios.

The improved heuristic significantly advanced deployment efficiency by resolving the optimality issues observed in large and congested networks. By targeting the point of maximum contention, the heuristic makes decisions that align better with the global optimum.

However, a small number of unsolved cases remain. Since the Brute Force solver found solutions for these cases, they represent feasible configurations that our heuristic cannot reach. These edge cases likely require complex, non-intuitive placements (e.g., shaping a flow at a point that does not directly impact the bottleneck but triggers a favorable domino effect) that our heuristic cannot easily predict. Covering these corner cases would likely require a significant increase in algorithmic complexity, for a small gain in optimality. Thus, we consider the refined heuristic as a good balance to improve the CBS deployment while minimizing complexity.

5.2.3 Schedulability Evaluation

The sensitivity analysis of delay bounds for shaped and unshaped flows under the improved and the original heuristic one has been conducted using the synthetic use cases when varying the network size and load. The results are shown in Figures 5.11 and 5.12, respectively.

In both figures, one can notice that the improved heuristic is not always on better for high-priority flows. Nevertheless, as size and load increase, the improved heuristic consistently outperformed the original. This is because at increased sizes and loads, the improved heuristic is able to deploy less CBS, thus reducing the impact on high-priority flows. These exceptions likely resulted from the order of device selection, which can affect the delays as placing a CBS downstream forces more contention upstream, leading to higher delays for shaped flows.

In addition, both heuristics are effective at ensuring the schedulability of unshaped flows as shown in both figures, reducing the number of unscheduled flows compared to NP-SP. This is true even for unsolved cases, where both heuristics still manage to have at worst 90% of the unshaped flows schedulable, compared to only 70% for NP-SP.

5.2.4 Scalability Evaluation

Finally, we assess the scalability of the improved framework through measuring the run time of the different methods for the synthetic use cases. Indeed, computational efficiency is a critical requirement for any design tool intended for large-scale industrial networks. Figures 5.13 and 5.14 plot the execution times for successful runs when varying network sizes and loads, respectively.

The obtained results are demonstrative. While the execution time of the Brute Force solver explodes exponentially with network size and load (reaching thousands of seconds), our heuristic maintains a stable execution time in the order of hundreds of milliseconds (400–500ms). Notably, the improved heuristic is often faster than the original one. By targeting the most critical contention points, the improved logic converges to a valid solution in fewer iterations. This confirms that our improved burstiness-aware heuristic successfully decouples optimization complexity from network scale, providing a solution that is three orders of magnitude faster than exact methods while maintaining a very low optimality gap.

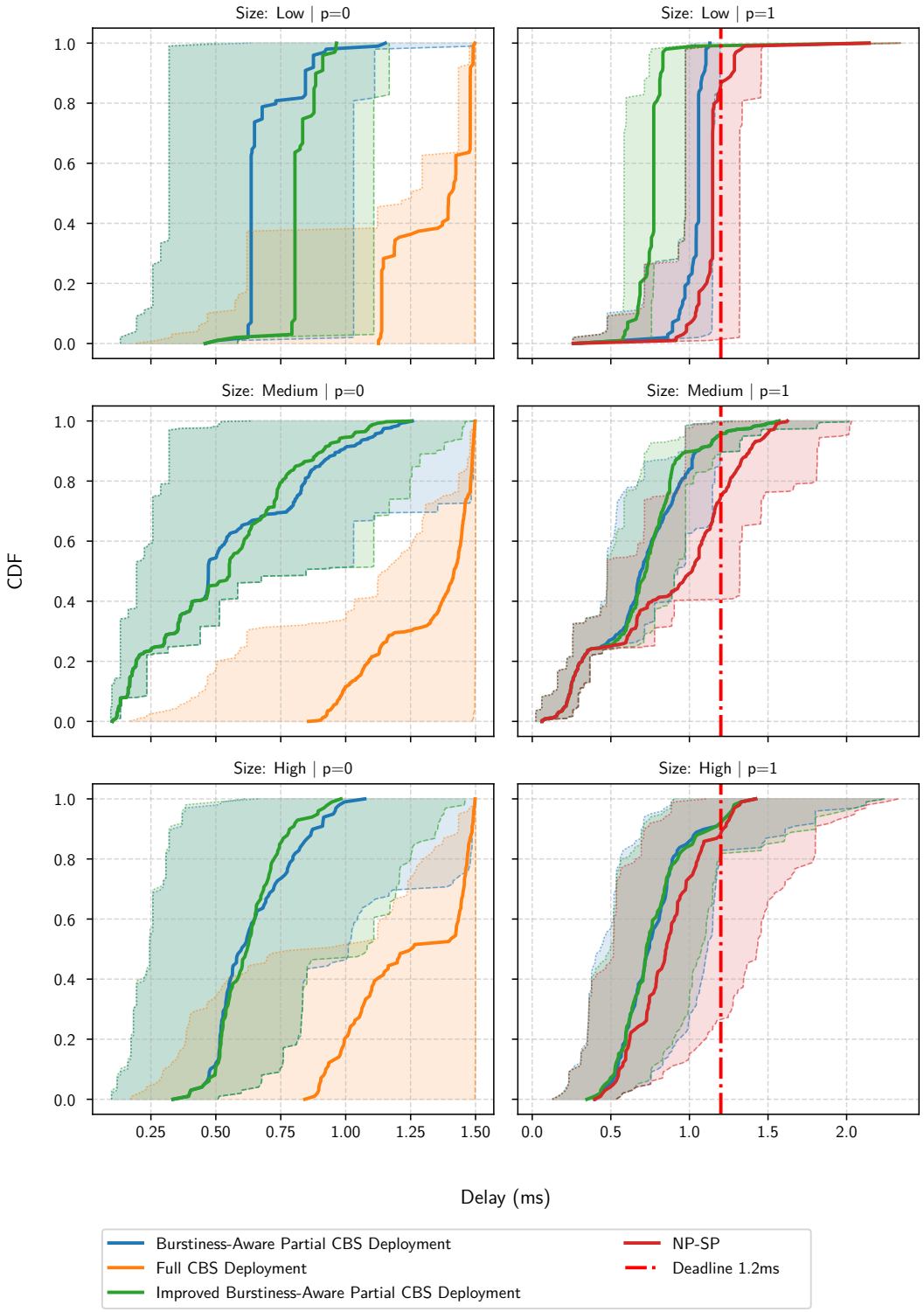


Figure 5.11: Delay bounds comparison (Variable Size) for Shaped Flows $p = 0$ (left column) and Unshaped Flows $p = 1$ (right column).

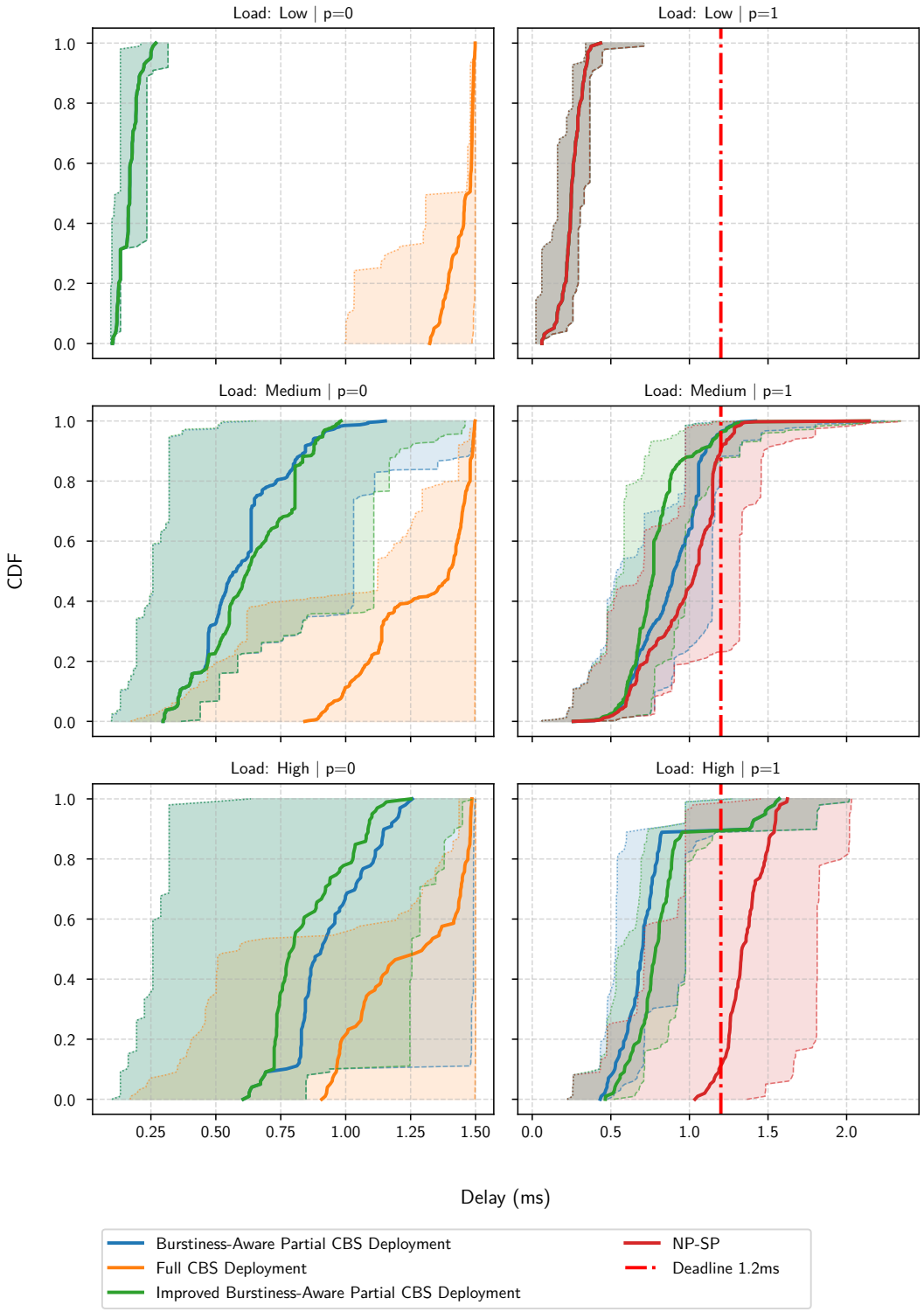


Figure 5.12: Delay bounds comparison (Variable Load) for Shaped Flows $p = 0$ (left column) and Unshaped Flows $p = 1$ (right column).

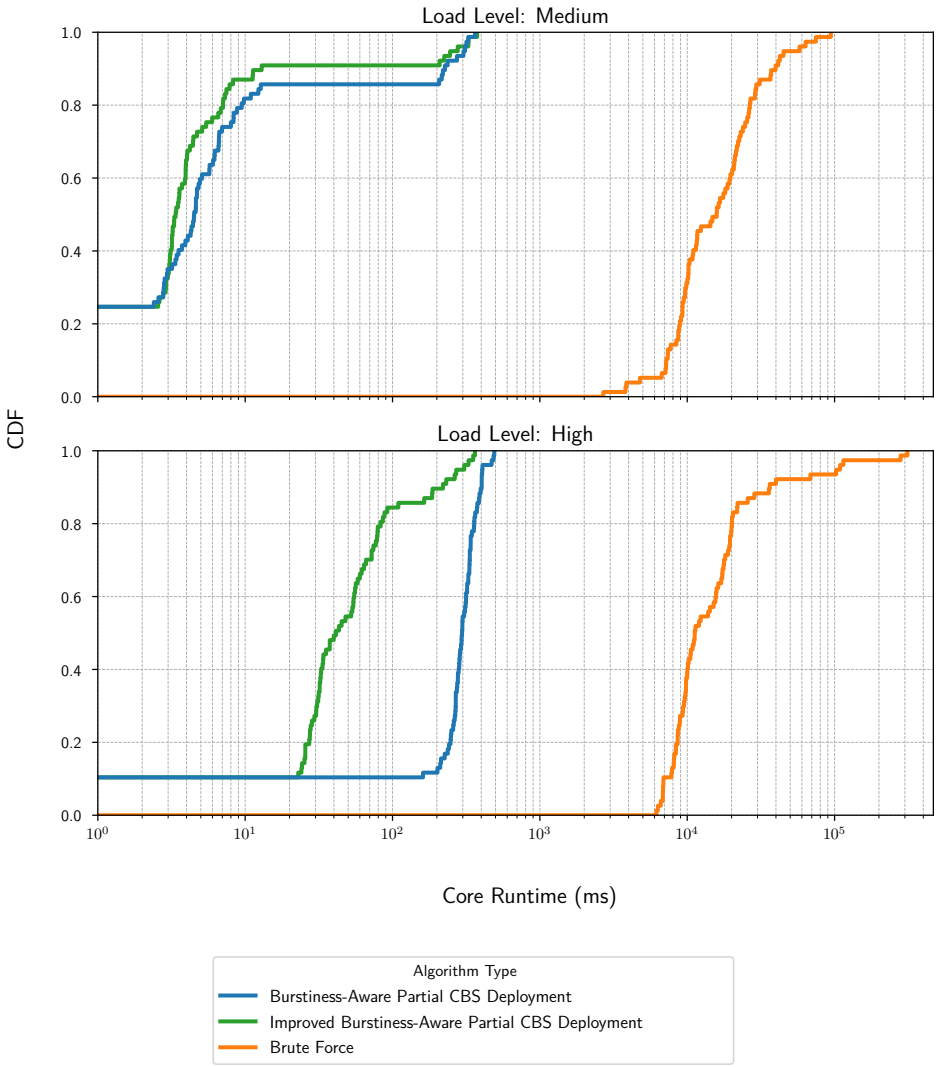


Figure 5.13: Execution time comparison (Variable Load)

5.2.5 Re-evaluation of the Industrial Use-case

Evaluating the industrial use-case presented in Section 3.5, we observe no measurable variation in overall performance between either improved approach, obtaining the same results overall. Table 5.3 shows the deployed CBS and their respective IdleSlope values.

We obtained identical results because, independent of the device selection heuristic, the most problematic flow requires placement of CBS on both $ZCP3 \rightarrow HPC$ and $HPC \rightarrow HPC_host$. The remaining unschedulabilities are solved placing a CBS in $HPC \rightarrow ZCP3$. The flows affected by the CBS in $HPC \rightarrow ZCP3$ are independent from the ones solved by the other two IdleSlopes. Thus, even though the device selection is done in the inverse order, the result is the same.

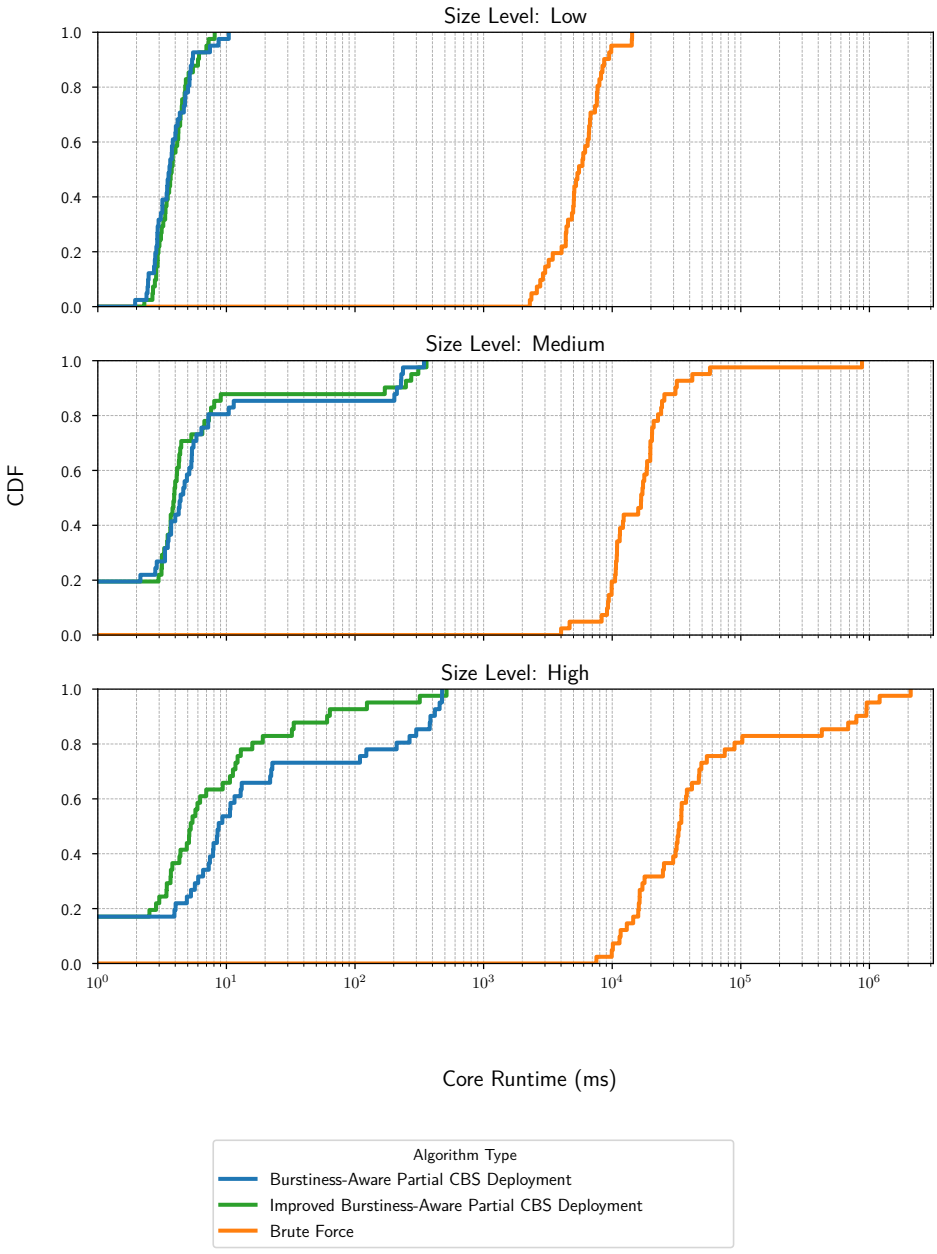


Figure 5.14: Execution time comparison (Variable Size)

5.3 Assessment of the FRER-Aware Partial CBS Deployment Framework

To assess the *FRER-Aware Partial CBS Deployment Framework* introduced in Chapter 4, we applied the same evaluation methodology, incorporating frame replication and elimination into the synthetic network scenarios.

Table 5.3: Deployed CBS for the Industrial Use-Case presented in Section 4.4 comparing the original burstiness-aware approach to its improved version

Output Port	**Burstiness-Aware Deployment IdleSlopes (Mbps)**	**Improved Burstiness-Aware Deployment IdleSlopes (Mbps)**
$ZCP3 \rightarrow HPC$	123	123
$HPC \rightarrow HPC_host$	123	123
$HPC \rightarrow ZCP3$	56	56

Since the improved burstiness-aware heuristic performs well in terms of optimality and scalability (as shown in previous section), we focus our evaluation on comparing the FRER-aware heuristic against this improved baseline. To isolate the impact of FRER-awareness, we add the burstiness-aware improvement to the FRER-aware framework, thus, whenever the *foi* selected does not have a PEF in its path, the device selected for CBS deployment is the one with the highest contention. To compare both heuristics, as in Chapter 4, the burstiness-aware framework computes the minimum IdleSlope considering the output arrival curve of PEFs, using Equation 4.16.

5.3.1 Optimality Evaluation

The optimality evaluation is conducted based on synthetic use-cases incorporating FRER that are generated following the scenarios described in Section 5.1. Among the 600 generated tests in this case, only 158 instances have a CBS deployment solution using Brute Force solver.

Then, we have applied our FRER-aware and FRER-blind heuristics on these 158 instances, and we have managed to solve only 37 instances which leads to a maximum success rate of $\frac{37}{158} = 23.4\%$.

This result highlights the challenge of handling FRER-induced burstiness, that is not fully addressed by simply improving the placement logic. The complexity introduced by frame replication and elimination appears to create contention patterns that are difficult to mitigate with the current heuristic approach.

To have a deeper idea about the quality of the obtained solution with our heuristics with reference to Brute Force method, we have gathered the deployed TSN devices under the different approaches for medium size and medium load networks, as these have the same configurations and are the ones who have the most solved instances. The results are shown in Figure 5.15.

As it can be observed in Figure 5.15, FRER-aware heuristic leads to either an equal or better CBS deployment solution than the FRER-blind one in situations where both heuristics are able to find a solution.

This comes by construction of the algorithm as specified in Section 4.4.4.

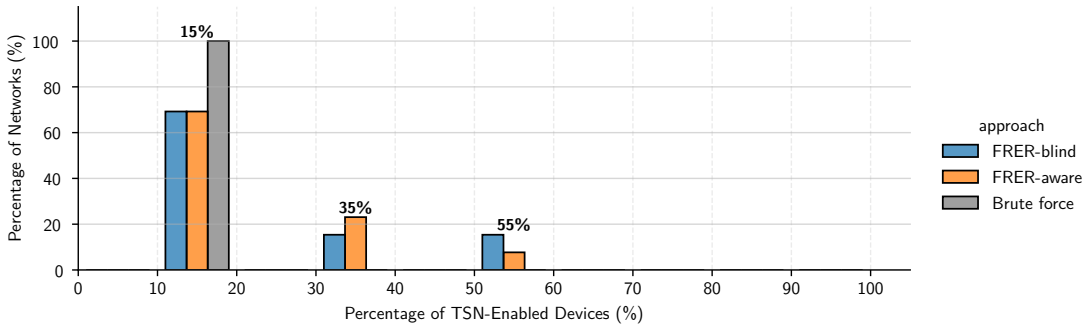


Figure 5.15: TSN Devices comparison in FRER networks

Afterwards, such an improvement has been quantified using the optimality gap from Equation 5.2. For the 37 solved instances, the average optimality gap for the FRER-aware heuristic is 7.7%, while the FRER-blind version has an average gap of 9.2%. This indicates that explicitly accounting for FRER-induced burstiness in the placement logic leads to more efficient CBS deployments, even if the overall success rate remains low. It also suggests that this refinement impact on the placement logic is low as the difference between the two gaps is only 1.5%.

The observed results concerning the optimality of both heuristics with reference to Brute Force have shown two main issues:

- the FRER-aware heuristics struggle to find solutions in the tested scenarios with a low success rate in comparison to Brute Force. This fact suggests that the current FRER-aware heuristic does not adequately address the complexities introduced by FRER mechanism;
- the optimality gap between FRER-blind and FRER-aware heuristic is low, which suggests that further refinement is needed.

A first investigation of these issues indicates that our relaxation of the problem in Section 4.3.1 may be too coarse, leading to over-estimations of burstiness and, consequently, overly conservative CBS placements.

A second assumption leading to this pessimism lies on the computation of too constrained local deadlines. Indeed, the considered heuristic for the end-to-end deadline repartition may limit our ability to find an IdleSlope close to the real minimum IdleSlope, especially in situations with high rates, which is the case of FRER-enabled networks.

These investigations helped us to identify a main area for improvement that consists in **Refining the IdleSlope computation**. Since the current relaxation does not allow us to reach the real minimum IdleSlope for ports implementing a PEF, we have developed a more accurate heuristic to compute local deadlines, and consequently a better IdleSlope considering FRER-induced burstiness that could lead to more efficient CBS placements.

5.3.2 Improved FRER-aware Partial CBS Deployment Framework

As stated, the main area of improvement is the computation of the minimum IdleSlope considering FRER-induced burstiness.

Consider Equation 4.16, reproduced below:

$$I_{op}^{p} \geq \frac{\sum_{f \in \mathcal{F}(op,p)} (b^f + \text{PEF}_{op}^{p} \cdot r^f \cdot J_{diff}^{f})}{D_{op}^{p} \cdot margin - \frac{\sum_{k=0}^{p-1} cred_{op}^{k,\min} - L_{op}^{>p,\max}}{\sum_{k=0}^{p-1} I_{op}^{k} - C}}$$

Where J_{diff}^{f} is computed as:

$$J_{diff}^{f} = \left(\max_{f_i \in \mathcal{I}_f} d_{\max}^{f_i} - \min_{f_j \in \mathcal{I}_f} d_{min}^{f_j} \right)$$

A possibility to improve the computation of the minimum IdleSlope is to refine the computation of the local deadline D_{op}^{p}. As explained in Chapters 3 and 4, the heuristic to compute the local deadline is based on the repartition of the end-to-end deadline taking into account the traffic rate within each output port *op* along the path. The idea of this heuristic is to allocate the longest deadline to the busiest output port.

However, in FRER-enabled networks, the interference patterns are more sophisticated due to replication and elimination, that may lead to a more constrained deadline partition.

In the particular case of FRER-enabled networks with FRER applied only once along the flow paths, we have introduced an improvement to compute less constrained local deadline for PEF output ports, which are the potential candidates for CBS placement in our FRER-aware heuristic.

The main modification concerns Equation 4.4 and it is as follows:

$$D_{op \ni PEF}^{p} = \min_{\forall f \in \mathcal{F}(op,p)} \left\{ \min_{\forall f_i \in \mathcal{I}^f | op \in \phi^{f_i}} \left[(D^f - \sum_{\forall op' \in \phi^f \setminus op} d_{op'}^{p} - d_{prop}) \right] \right\} \tag{5.3}$$

The idea is to consider that the delay bounds within all crossed output ports are computed under NP-SP except the one within the port implementing the PEF ($op \ni PEF$). Hence, the allocated local deadline to the PEF port is the left slack after substracting all the computed delay bounds from the end-to-end deadline.

This new heuristic allows ignoring the repartition based on rates and leads to a less constrained local deadline for the port with PEF.

We have included such an improvement of local deadline computation within the FRER-

aware framework, that we call improved FRER-aware framework. Afterwards, we have launched the improved FRER-aware heuristic on the 158 network instances, where the Brute Force solver found solutions, and the success rate is 100%. Such a notable result validates our intuition that a refined local deadline and consequently IdleSlope computation effectively addresses the previous limitations of the original FRER-aware heuristic and leads to feasible configurations across all tested network scenarios.

The comparison of the deployed TSN devices under the different approaches including the improved FRER-aware heuristic for all tested networks of medium size and load has been conducted and results are shown in Figure 5.16.

When compared against the FRER-blind and original FRER-aware frameworks, the improved FRER-aware framework consistently requires fewer TSN devices for CBS deployment across all tested scenarios. Moreover, in up to 80% of the scenarios, the improved FRER-aware heuristic converges to the global optimum found under the Brute Force method which validates its notable optimality.

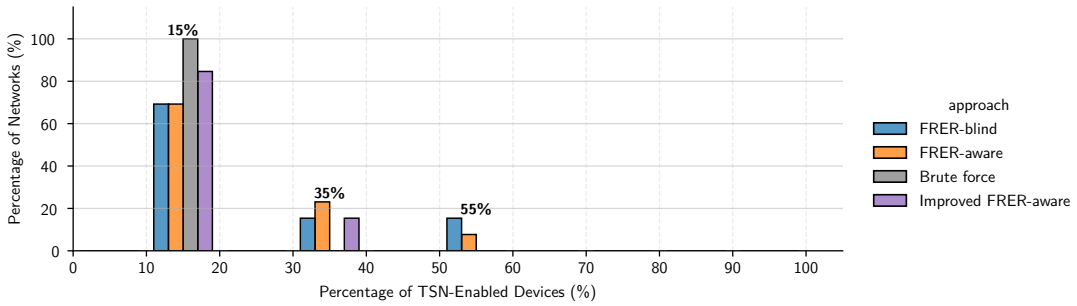


Figure 5.16: Improved FRER-aware TSN Devices comparison

To further assess the optimality of the improved heuristic, we have computed the optimality gap of the different heuristics as shown in Figure 5.17.

It is clear that the optimality gap of the improved FRER-aware heuristic is consistently lower than both previous versions across all tested scenarios of medium size and load. Particularly, the average optimality gap of the improved FRER-aware heuristic in this situation is around 3.8%, significantly lower than both the FRER-blind (9.2%) and original FRER-aware (7.7%) heuristics. This result demonstrates that the refined IdleSlope computation not only enables finding more feasible solutions but also leads to more efficient CBS deployments, closely approximating the optimal configuration.

Now that we can solve all the test- cases, we directly compare the improved FRER-aware framework against the Brute Force solver for all network instances. The results are shown in Figures 5.18 and 5.19 when varying network load and size, respectively.

As it can be noticed, the improved FRER-aware heuristic represents a step forward, effectively resolving the feasibility issues observed in FRER-enabled networks. By refining the IdleSlope computation and tightening the deadline repartitioning, the heuristic ensures

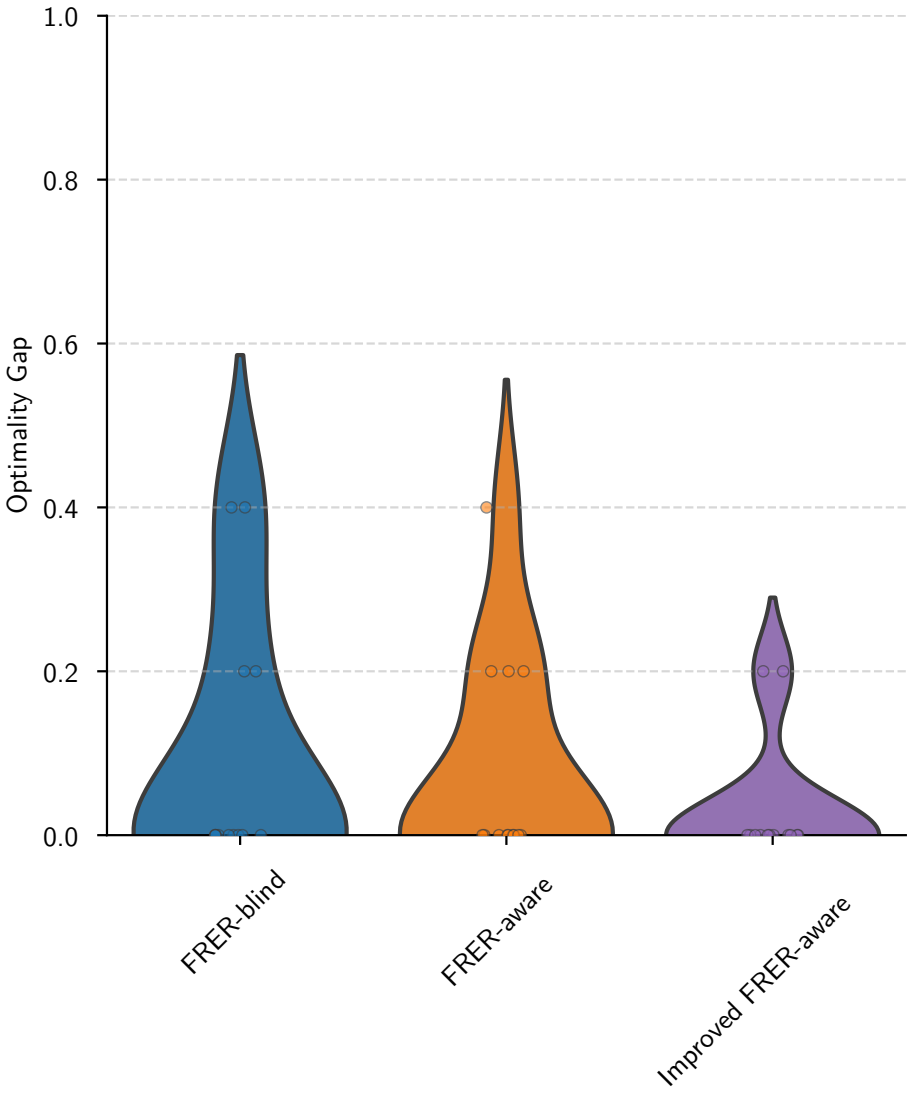


Figure 5.17: Improved FRER-aware optimality gap

that a valid solution is found whenever one exists. However, unlike in the case of FRER-disabled networks, the optimality gap tends to widen significantly as network load and size increase. Since the Brute Force solver consistently finds solutions with fewer devices in these scenarios, this indicates that our heuristic, while safe, is still conservative.

This suggests that further refinements may be needed to enhance the heuristic's ability to improve the CBS placements and the number of total TSN devices. Effectively, the estimation of the burstiness-increase due to FRER is still pessimistic as we do not account for the complete Corollary 1 of [92], shown in Equation 4.1, which could lead to more constrained IdleSlope computations. This could be subject of future work.

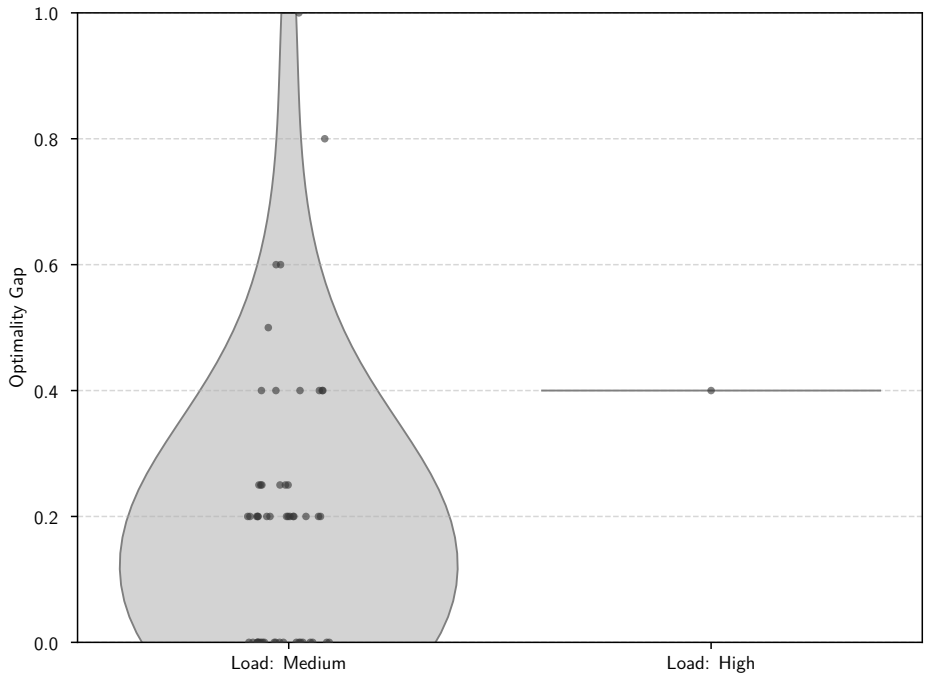


Figure 5.18: Improved FRER-Aware Optimality Gap (Var. Size)

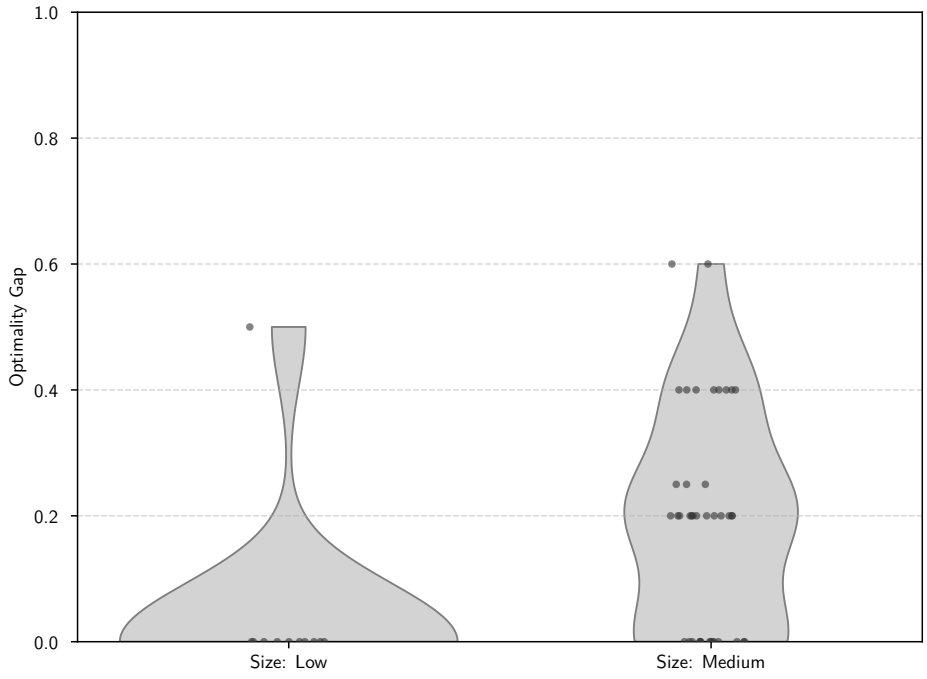


Figure 5.19: Improved FRER-Aware Optimality Gap (Var. Load)

5.3.3 Schedulability Evaluation

In this section, we analyze the schedulability of shaped and unshaped traffic under the different heuristics for the synthetic use cases when enabling FRER and varying the network load and size. We consider only the network instances where the heuristics converge to a feasible solution.

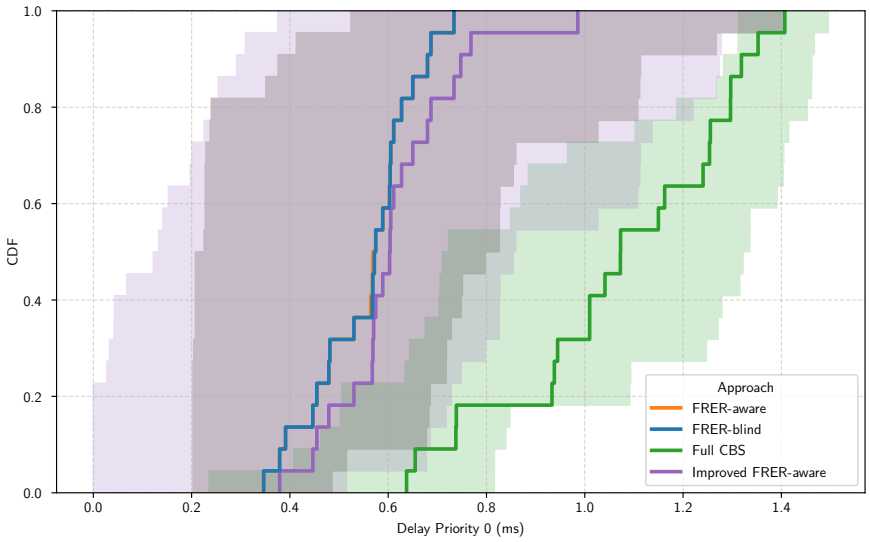


Figure 5.20: Improved FRER-aware Shaped delay (Var. Size, $p = 0$)

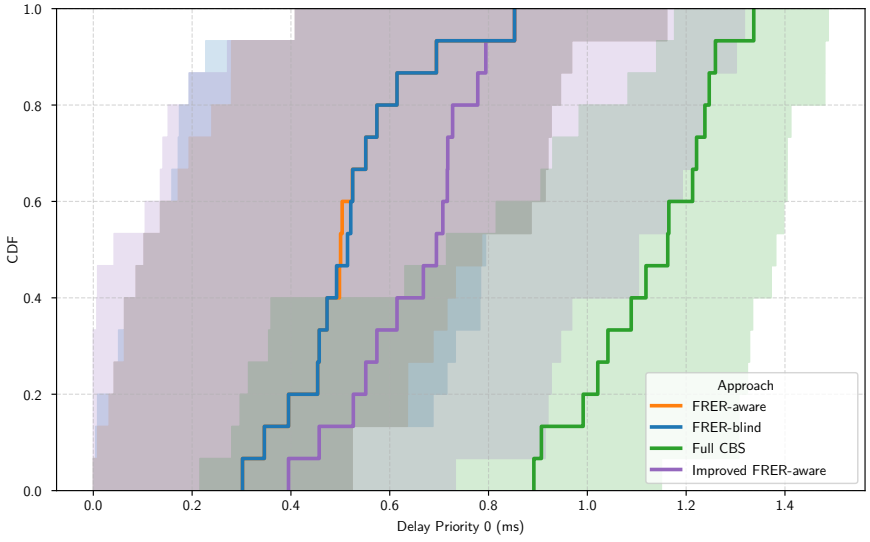


Figure 5.21: Improved FRER-aware Shaped delay (Var. Load, $p = 0$)

For shaped flows (p=0), Figure 5.20 for size variation and Figure 5.21 for load variation show that the improved FRER-aware heuristic has higher delay bounds for shaped flows compared to the other two heuristics. Since the improved FRER-aware heuristic is able to

use a tighter IdleSlope, it is able to exploit better CBS deployments. Nevertheless, due to this tighter value, the impact on shaped traffic is more important than on either the FRER-blind or original FRER-aware heuristics.

Regarding the impact on the unshaped flows, this can be observed in Figure 5.23 for load variation and Figure 5.22 for size variation, respectively.

The results show a less observable impact. This is because the network instances where all heuristic find a solution are often characterized by a low amount of unscheduled flows. Thus the difference between the heuristics is small, as the placed CBS impact a low amount of unshaped flows. However, it is worth noting that all heuristics solve the schedulability issues observed under NP-SP.

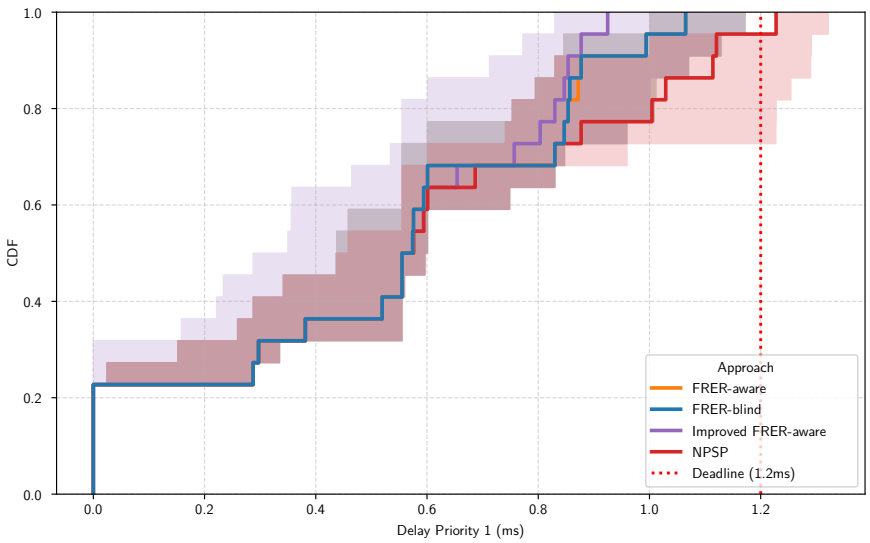


Figure 5.22: Improved FRER-aware Unshaped delay (Var. Size, $p = 1$)

5.3.4 Scalability Evaluation

Finally, we assess the scalability of the different heuristics through computing the execution times for the synthetic cases enabling FRER when varying the size and load. These execution times are compared to the Brute Force execution time to highlight the efficiency of our proposed framework.

Figures 5.24 and 5.25 show the execution times for successful runs when varying the network load and size, respectively.

Both Figures show that the execution times regardless of the heuristic are much lower than the Brute force method which validates their efficiency. Moreover, we can notice that although there is a slight increase in execution time compared to the other heuristics, the improved FRER-aware framework maintains computational efficiency suitable for large-scale industrial networks. For instance, it remains stable in the order of tens of milliseconds, where

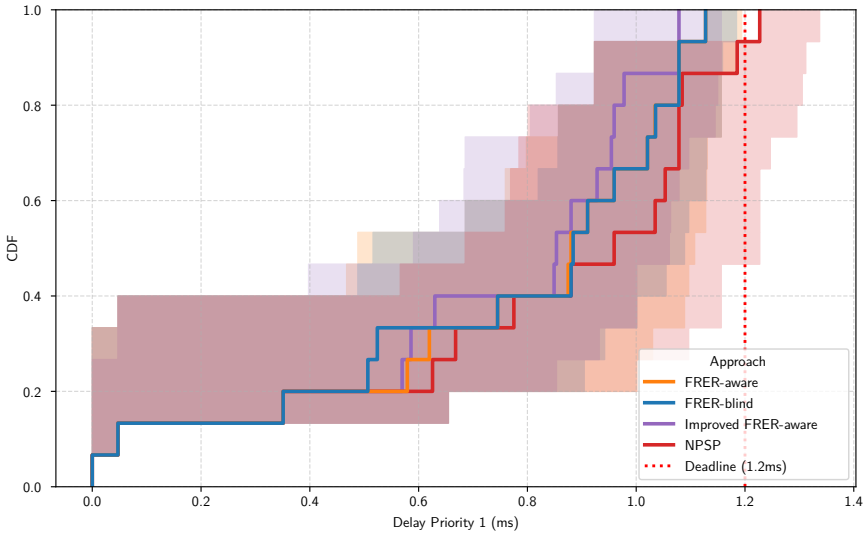


Figure 5.23: Improved FRER-aware Unshaped delay (Var. Load, $p = 1$)

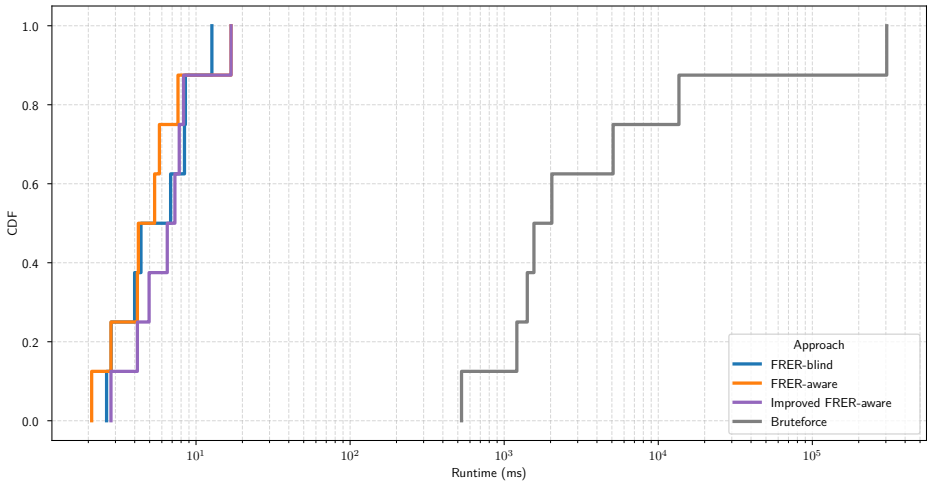


Figure 5.24: Improved FRER-aware execution time (Var. Load)

at worse it reaches 24ms of execution time. This is coherent with the increase in the quality of the solution, which is more expensive in terms of execution time.

5.3.5 Re-evaluation of the Industrial Use-case

Similar as in the previous section, we re-evaluate the Industrial Use-case adapted with FRER from Section 4.5. Here, we obtained the same CBS deployments, but with updated IdleSlope values. Table 5.4 shows the different IdleSlope values obtained per approach.

The main observation being that the ports with a PEF implemented ($HPC \rightarrow HPC_Host$ and $HPC_AD \rightarrow HPC_AD_Host$) have a lower or, at most, the same IdleSlope value as

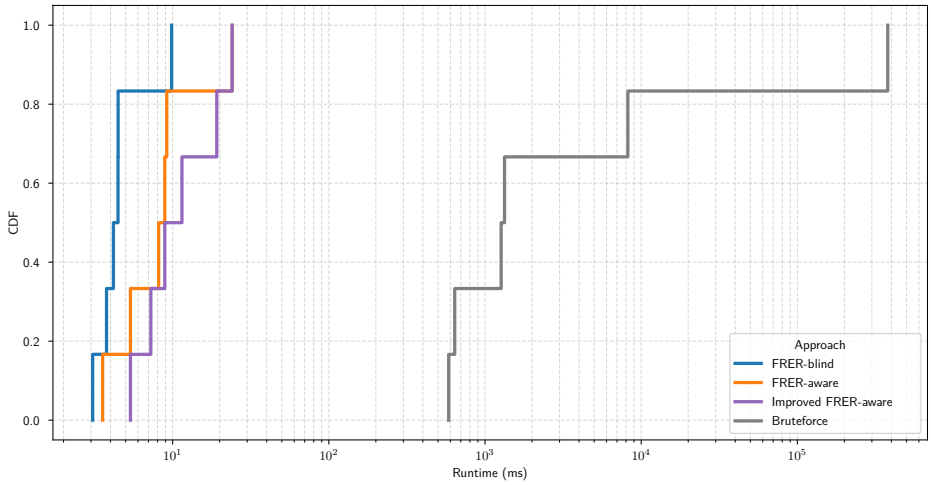


Figure 5.25: Improved FRER-aware execution time (Var. Size)

Table 5.4: Deployed CBS Parameters under FRER-aware and its improved version for the Industrial Use-Case of Section 4.5

Output port	**FRER-Aware Partial CBS (Mbps)**	**Improved FRER-Aware Partial CBS (Mbps)**
$HPC \rightarrow HPC_Host$	151	151
$HPC \rightarrow ZCP1$	170	213
$HPC \rightarrow ZCP2$	144	181
$HPC \rightarrow ZCP3$	58	73
$HPC_AD \rightarrow HPC_AD_Host$	170	130

in the original version of the FRER-aware framework. The other IdleSlope values are higher for the Improved FRER-aware framework. This is due to the fact the the originally placed CBS forced the need for higher IdleSlopes to maintain the schedulability of shaped flows.

5.4 Conclusions

This chapter presented a thorough validation of our Partial CBS Deployment Framework, utilizing a custom-built Brute Force solver and a synthetic network generator to benchmark our heuristics against optimal solutions. This extensive evaluation allowed us to identify some limitations in our initial approaches, leading to significant algorithmic improvements while also highlighting open challenges regarding FRER-enabled networks.

In FRER-disabled case, our initial analysis revealed that while the original *Burstiness-Aware* heuristic outperformed Full CBS deployments, its optimality gap in large and high-load networks was high. This issue was traced to the *first encounter* logic, which prematurely reduced the search space by targeting the earliest, rather than the most critical, point of contention.

By shifting the heuristic strategy to target the *highest contention* point, specifically where the aggregate rate of higher-priority unshaped traffic is maximized, we successfully handled this limitation and proposed an improved burstiness-aware heuristic. The resulting *Improved Partial CBS Deployment Framework* demonstrated:

- **Near-Optimal Performance:** The optimality gap was drastically reduced, falling from 14% to 5% for large networks and from 57% to 1% in high-load scenarios.
- **Scalability:** The framework maintained stable execution times in the order of hundreds of milliseconds, successfully decoupling computational complexity from network scale, whereas exact methods suffered exponential time explosions.
- **Efficiency:** It achieved these results while maintaining safe delay bounds for both shaped and unshaped traffic, validating the cost-efficiency of the partial deployment strategy.

In FRER-enabled cases, the initial evaluation of the *FRER-Aware* framework revealed that standard heuristic approaches struggled with the complexity of frame replication, resulting in low success rates to find a deployment solution due to excessive pessimism in the IdleSlope calculation. By introducing the *Improved FRER-Aware Partial CBS Deployment* heuristic, which utilizes a refined IdleSlope computation thanks to a tighter deadline repartitioning strategy, we effectively resolved these feasibility issues. The improved heuristic achieved a 100% success rate across all network instances where a solution was known to exist, demonstrating that accurate mathematical modeling of FRER-induced burstiness is the key to unlocking valid configurations in reliability-constrained networks.

Furthermore, the improved FRER-aware heuristic significantly enhanced solution quality, achieving an average optimality gap of just 3.8% in medium-load scenarios, surpassing both the FRER-blind and original FRER-aware versions. While the optimality gap still exhibits a tendency to widen slightly under extreme load and size variations, the framework maintains the computational efficiency required for large-scale industrial design with reference to Brute Force method.

Hence, the results presented in this chapter confirm that the proposed framework provides a sub-optimal, scalable and cost-efficient method for deploying CBS in complex, high-reliability time-sensitive networks.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The transition from legacy Ethernet to TSN represents a paradigm shift in real-time communications, offering the determinism required for real-time applications in diverse critical domains. However, as identified in the introduction to this document, the assumption of a full TSN deployment creates significant barriers regarding cost and interoperability with existing infrastructure. This dissertation addressed these challenges by proposing a comprehensive framework for the *Partial Deployment* of the CBS.

The primary contribution of this work is the development of a cost-efficient methodology that enables the strategic coexistence of TSN-capable switches implementing CBS and legacy devices. By leveraging the NC framework, we demonstrated that meeting the scheduling requirement (the main requirement of real-time networks) does not require a complete hardware overhaul, but rather a targeted intervention at critical points of contention.

In Chapter 3, we introduced the concept of *Burstiness-Aware* deployment. It was demonstrated that the primary driver for schedulability violations in mixed-criticality networks where the high-priority class is bursty is the unchecked burstiness accumulated by high-priority flows traversing legacy nodes. We proposed a heuristic capable of identifying these bottleneck devices and configuring CBS instances with the minimum necessary `IdleSlope` to restore schedulability without starving lower-priority traffic.

In Chapter 4, we extended this system model to incorporate high-reliability constraints, specifically the FRER mechanism. Our analysis revealed that the reliability provided by FRER comes at the cost of protocol-induced burstiness generated at elimination points. We demonstrated that a standard, *FRER-blind*, deployment is inefficient in such scenarios. Consequently, we developed a *FRER-aware* framework that targets CBS placement specifically at PEF nodes, effectively neutralizing the adverse effects of redundancy on network latency.

Finally, in Chapter 5, we rigorously evaluated the efficiency of our heuristics against an optimal brute-force baseline. While the initial heuristic showed limitations in high-load and large-scale scenarios, the development of the *Improved Burstiness-Aware* framework, which targets the point of *highest* contention rather than the *first*, successfully reduced the optimality gap to approximately 7% in high-load scenarios and 26% for large networks. The improved framework remains scalable, requiring at most approximately 0.5 seconds, instead of the

more than 1.5 hours required by an exact solver. This confirmed that our framework offers a scalable, near-optimal solution for designing heterogeneous real-time networks.

Moreover, we identified that both the Improved Burstiness-Aware and FRER-Aware heuristics struggled to converge in FRER-enabled scenarios. To address this, we proposed an *Improved FRER-Aware* approach that utilizes a refined method for computing tighter `IdleSlope` values. Although this method does not guarantee optimality, it significantly enhances success rates compared to previous strategies, marking a substantial advancement in the practical deployment of CBS in complex network environments.

The findings of this dissertation directly address the research questions established in Chapter 2:

R.Q.1: Where can CBS be effectively deployed to leverage its benefits, while maintaining interoperability with legacy Ethernet devices? We determined that CBS is most effective not when deployed ubiquitously, but when placed at specific boundaries where legacy traffic characteristics jeopardize critical medium-priority flows. Specifically, our results show that CBS should be deployed at the point of maximum accumulated contention (as demonstrated by the Improved Framework in Chapter 5) or immediately downstream of FRER elimination functions (as shown in Chapter 4). These locations provide the highest return on investment regarding burstiness mitigation.

R.Q.2: When is CBS an effective solution to improve network performance, considering the different traffic patterns and network topologies? Our validation scenarios indicate that Partial CBS is a viable solution when the network fabric is heterogeneous and the primary inhibitor to schedulability is the delay caused by bursty high-priority traffic. Furthermore, we showed that CBS is particularly necessary when redundancy mechanisms (FRER) are active, as the elimination process artificially increases frame clustering, requiring immediate reshaping to maintain downstream determinism.

R.Q.3: How do we achieve CBS integration into a TSN network, and how can CBS parameters be optimally computed? We achieved integration through a heuristic-based framework validated by NC, where the idea is to deploy CBS in points of interest. We have identified potential points of interest and have validated them through industrial use cases and synthetic test cases, which revealed their effectiveness. Regarding configuration, we concluded that the optimal strategy is to compute the minimum `IdleSlope` required to satisfy the stability condition and the local deadline constraints. This *minimum-necessary* configuration strategy minimizes the blocking effect on the shaped class itself while preserving the maximum possible bandwidth for the unshaped, medium-priority traffic that relies on legacy scheduling.

To conclude, the Partial CBS Deployment framework presented here represents a significant step towards more flexible and cost-effective TSN network designs. By addressing the challenges of interoperability and cost, this work paves the way for broader adoption of TSN technologies in diverse industrial applications.

6.2 Future Work

For future research, we have identified the following potential research directions.

Further improvements for Partial CBS deployment

- **Optimality Improvement:** Although the *Improved* heuristic significantly closed the optimality gap, it remains a non-exact approach. As discussed in Chapter 5, there exist corner cases involving complex, non-intuitive placements (e.g., shaping a flow upstream to trigger a favorable effect downstream) that the current logic cannot systematically identify. Moreover, with FRER integration the challenge increases in complexity as we have identified a strategy to perform partial CBS deployment in these contexts, but it is not always optimal. The key idea lies on reducing the pessimism due to the problem linearization. Some potential solutions consist of considering a tighter arrival curve at the output of PEF or a better heuristic for local deadline computation to further improve IdleSlope computation.

- **Extension of the optimization problem:** The optimization problem formulating the partial CBS deployment can be extended by incorporating different objective functions or additional constraints. The current optimization problem is formulated to find the *minimum* number of TSN devices required for schedulability. While cost-effective, this does not reflect all industrial design objectives. A network designer may operate under a fixed budget that allows for a specific ratio of legacy-to-TSN switches, or they may wish to retain a specific percentage of legacy hardware for compatibility reasons. The current approach does not allow for a target ratio to be set as a constraint. Consequently, future work could focus on inverting the optimization logic: maximizing the schedulability margin or traffic load capacity given a fixed, pre-defined cardinality of TSN-enabled devices.

  Furthermore, while the framework successfully identifies optimal deployment locations for CBS, it does not assume legacy devices as a constraint in the optimization process. In real-world scenarios, certain legacy switches may be non-upgradable due to physical or budgetary constraints, which could limit the applicability of the proposed solutions.

  Another potential challenge would be to ignore the priority constraint where CBS cannot be placed on a priority for which any priority higher does not implement CBS. Future analysis of this mathematically viable relaxation could reveal novel insights into CBS placement dynamics.

  Lastly, we have considered a redundancy scheme where only two replicas were generated per flow. An interesting perspective would be to analyze a network model where $n > 2$ replicas are considered. This would affect CBS placement as we have already seen in this dissertation that the added rate and burstiness are not necessarily easily mitigated with CBS.

- **Scalability improvement** to enable on-line configuration, one can further improve the

execution time of the framework. This will allow adaptive CBS deployment strategies that can respond to changing network conditions, such as varying traffic loads or the addition/removal of legacy devices.

Generalization of the framework to other TSN mechanisms

- **Use of different TSN shapers** One can extend the current framework to consider additional TSN shapers that interplay with CBS, such as TAS, CQF, ATS. This would provide a more holistic optimization approach for mixed TSN networks. An interesting challenge would be to consider multiple shaping mechanisms in the same output port, where there is one per priority.
- **Extended use of the FRER mechanism:** We have considered in our framework the common use of FRER with one replication/elimination segment along a flow path. However, a general use of FRER may induce many segments which increases the reliability level of the network. However, this general use may increase the complexity of finding the optimal CBS placement, as the growing number of elimination points expands the solution space.

Appendix A

Network Calculus

Contents

A.1 Mathematical Background: The (min, +) Algebra 122
A.1.1 Min-Plus Convolution ($\otimes$) 122
A.1.2 Min-Plus Deconvolution ($\oslash$) 122
A.2 Data Modeling and Core Concepts 122
A.2.1 Data Flow and Server Model 123
A.2.2 Backlog and Virtual Delay 123
A.3 Arrival Curves 123
A.3.1 Definition 123
A.3.2 The Leaky-Bucket Curve ($\gamma_{r,b}$) 124
A.4 Service Curves 124
A.4.1 Definition 124
A.4.2 The Rate-Latency Curve ($\beta_{R,T}$) 124
A.4.3 Strict Service Curve 125
A.5 NC Basics: The Single Node 125
A.6 Network Analysis: Multiple Nodes 126
A.6.1 Concatenation of Servers (Tandem Networks) 126
A.6.2 Flow Multiplexing and Residual Service 126
A.6.3 Residual Service for Non-Preemptive Static Priority (NP-SP) 126
A.6.4 Modular Analysis (TFA) 127

NC is a theory of deterministic queuing systems [51]. It provides a mathematical framework for deriving worst-case performance bounds in communication networks, in contrast to traditional queuing theory which typically provides stochastic or average-case results. This methodology is particularly suited for systems requiring hard performance guarantees, such as real-time embedded networks.

This annex provides a formal introduction to the core concepts of deterministic NC. It begins with the essential (min, +) algebra, defines the core data models, then specifies arrival and service curves. Subsequently, it presents the fundamental theorems for calculating single-node bounds, and concludes with methods for analyzing networked systems.

A.1 Mathematical Background: The (min, +) Algebra

NC is built upon the algebraic structure $(\mathbb{R} \cup \{+\infty\}, \inf, +)$, often referred to as the (min, +) dioid. In this algebra, the standard addition operation is replaced by *minimum* (or *infimum*), and standard multiplication is replaced by *addition* [51, Ch. 3]. This structure defines two key operators on functions that are fundamental to the theory.

A.1.1 Min-Plus Convolution ($\otimes$)

The min-plus convolution of two functions f and g is the (min, +) equivalent of standard convolution.

Definition A.1 (Min-Plus Convolution)
[51, Sec. 3.1.6]

$$(f \otimes g)(t) = \inf_{0 \le s \le t} \{f(s) + g(t-s)\}$$

This operator is used to model the cumulative output of a system, $D(t)$, and to compute the equivalent service of a tandem of network elements.

A.1.2 Min-Plus Deconvolution ($\oslash$)

The (min, +) complement operation to convolution is deconvolution.

Definition A.2 (Min-Plus Deconvolution)
[51, Sec. 3.1.9]

$$(f \oslash g)(t) = \sup_{s \ge 0} \{f(t+s) - g(s)\}$$

This operator is the primary tool for calculating performance bounds. It is used to determine the arrival curve of a flow after it has passed through a server and to compute the residual service in a shared system.

A.2 Data Modeling and Core Concepts

The core of NC lies in modeling data flows and the network servers that handle them using non-negative, non-decreasing functions of time $t \in \mathbb{R}^+$.

A.2.1 Data Flow and Server Model

A data flow is described by its *cumulative arrival function*, $A(t)$, which represents the total amount of data (e.g., in bits) from the flow that has arrived at a system in the time interval $[0, t]$.

A network server (e.g., a scheduler or an entire network) is modeled as a system that transforms an arrival function $A(t)$ into a *cumulative departure function* $D(t)$. For a lossless system, $D(t)$ represents the total amount of data from the flow that has departed the system in the interval $[0, t]$. By definition, $D(t) \leq A(t)$ for all t.

A.2.2 Backlog and Virtual Delay

From these two cumulative functions, the primary performance metrics are defined [51, Sec. 1.1.2].

Definition A.3 (Backlog)
The backlog $back(t)$ *in a system at time* t *is the amount of data that has arrived but not yet departed:*

$$back(t) = A(t) - D(t)$$

The maximum backlog, $\sup_{t \geq 0}\{back(t)\}$, represents the worst-case buffer occupancy.

Definition A.4 (Virtual Delay)
The virtual delay $d(t)$ *at time* t *is the time a bit of data arriving at time* t *would have to wait until it is served, assuming a FIFO service order.*

$$d(t) = \inf\{\tau \geq 0 \mid A(t) \leq D(t + \tau)\}$$

The maximum virtual delay, $\sup_{t \geq 0}\{d(t)\}$, represents the worst-case end-to-end latency for the flow.

A.3 Arrival Curves

A.3.1 Definition

To perform worst-case analysis, the exact function $A(t)$ is bounded by an *arrival curve*, $\alpha(t)$.

Definition A.5 (Arrival Curve)
[51, Def. 1.2.1] A flow A *is constrained by an arrival curve* α *if and only if for all* $0 \leq s \leq t$*:*

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

This inequality states that the data arriving in any time interval $(s, t]$ is bounded by α evaluated at the duration of that interval. An equivalent definition using (min, +) convolution is: $A \leq A \otimes \alpha$ [51, Lemma 1.2.3].

A.3.2 The Leaky-Bucket Curve ($\gamma_{r,b}$)

A prevalent model for an arrival curve is the *leaky-bucket* curve. It models a flow constrained by a token bucket with a token generation rate r and a bucket size (or burst tolerance) b.

Definition A.6 (Leaky-Bucket Curve)
The leaky-bucket arrival curve $\gamma_{r,b}$ is defined as:

$$\gamma_{r,b}(t) = b + r \cdot t, \quad \textit{for } t \geq 0$$

Here, b represents the maximum data burst that can arrive, and r represents the long-term average rate.

A.4 Service Curves

A.4.1 Definition

A network server is modeled by a *service curve*, β, which provides a minimum performance guarantee.

Definition A.7 (Service Curve)
[51, Def. 1.3.1] A system S offers a service curve β to a flow if, for all $t \geq 0$, its arrival and departure functions satisfy:

$$D(t) \geq (A \otimes \beta)(t)$$

This relation provides a lower bound on the service a flow will ever receive. It guarantees that for any time t, there exists some time $s \leq t$ such that the output at time t is at least the input at time s plus the minimum service $\beta(t - s)$ guaranteed during that backlogged interval.

A.4.2 The Rate-Latency Curve ($\beta_{R,T}$)

The most common service curve model is the *rate-latency* service curve. This models a server that guarantees a minimum rate R after incurring a maximum latency T (e.g., scheduling or processing delay).

Definition A.8 (Rate-Latency Curve)
[51, Def. 1.3.2] The rate-latency service curve $\beta_{R,T}$ is defined as:

$$\beta_{R,T}(t) = R \cdot [t-T]^+ = \max(0, R \cdot (t-T))$$

A.4.3 Strict Service Curve

For results, a stronger service guarantee is often required.

Definition A.9 (Strict Service Curve)
[51, Def. 1.3.2] A system offers a strict service curve β^S *if, during* any *backlogged period of duration $t-s$, the output of the flow is at least $\beta^S(t-s)$. Thus a strict service curve satisfies*

$$\forall \textit{backlogged period }]s,t], \quad D(t) - D(s) \geq \beta^S(t-s)$$

Where the *backlogged period* $]s,t]$ is defined as any interval where the backlog is strictly positive.

This is a stronger guarantee than the standard service curve, which only applies from the start of a backlogged period. A system offering a strict service curve β^S also offers a standard service curve $\beta = \beta^S$.

A.5 NC Basics: The Single Node

Given a flow constrained by an arrival curve α traversing a system offering a service curve β, NC provides the following theorems for worst-case performance bounds [51, Ch. 1.4].

Theorem A.1 (Backlog Bound)
[51, Thm. 1.4.1] The maximum backlog $back(t)$ experienced by the flow is bounded by the maximum vertical deviation between the arrival and service curves:

$$back(t) \leq \sup_{s \geq 0}\{\alpha(s) - \beta(s)\}$$

Theorem A.2 (Delay Bound)
[51, Thm. 1.4.2] The maximum virtual delay $d(t)$ experienced by the flow is bounded by the maximum horizontal deviation between the arrival and service curves:

$$d(t) \leq \sup_{s \geq 0}\{\inf\{\tau \geq 0 : \alpha(s) \leq \beta(s+\tau)\}\}$$

Theorem A.3 (Output Arrival Curve)
[51, Thm. 1.4.3] The output flow $D(t)$ is constrained by a new arrival curve, $\alpha^*(t)$, which is

derived using (min, +) deconvolution:

$$\alpha^*(t) = (\alpha \oslash \beta)(t) = \sup_{s \geq 0}\{\alpha(t+s) - \beta(s)\}$$

This output curve α^* is central to network analysis, as it describes the traffic that will enter the next network element.

A.6 Network Analysis: Multiple Nodes

The single-node models can be extended to analyze end-to-end network paths.

A.6.1 Concatenation of Servers (Tandem Networks)

When a flow traverses multiple servers $S_1, \ldots, S_n$ in sequence (i.e. a tandem), where each server S_k offers a service curve β_k, a single service curve can be found to represent the entire end-to-end path.

Theorem A.4 (Concatenation)
[51, Thm. 1.4.6] A tandem of n servers $S_1, \ldots, S_n$ offering individual service curves $\beta_1, \ldots, \beta_n$ is equivalent to a single server offering an end-to-end service curve β_{e2e}:

$$\beta_{e2e} = \beta_1 \otimes \beta_2 \otimes \ldots \otimes \beta_n$$

This result is known as the *Pay Bursts Only Once (PBOO)* principle. For example, the convolution of n rate-latency servers ($\beta_k = \beta_{R_k,T_k}$) results in $\beta_{e2e} = \beta_{R_{e2e},T_{e2e}}$ where $R_{e2e} = \min(R_k)$ and $T_{e2e} = \sum T_k$. This demonstrates that while latencies (T_k) add, the network only has to provision buffer for the flow's burstiness based on the bottleneck rate (R_{e2e}), not at every hop.

A.6.2 Flow Multiplexing and Residual Service

When multiple flows share a single server, the service guaranteed to any single flow depends on the server's scheduling policy and the traffic from all other interfering flows. The service curve left over for a specific flow is its *residual service curve.*

A.6.3 Residual Service for Non-Preemptive Static Priority (NP-SP)

Theorem A.5 (Residual Service Curve for NP-SP)
[103, Thm. 8.3] Consider a system with strict Service Curve β^S and m flows $f_1, f_2, \ldots, f_m$.

The maximum frame length of f_i is $L^{i,\max}$ and f_i is α_i-constrained. Flows are scheduled by NP-SP policy. For each $i \in \{1, \ldots, m\}$, the strict Service Curve of f_i is[1]*:*

$$\left(\beta^S - \sum_{j<i} \alpha_j - \max_{k \geq i} L^{k,\max}\right)^+_{\uparrow}$$

A.6.4 Modular Analysis (TFA)

A common method for network analysis is the modular, node-by-node approach, often termed *TFA* [65]. This method analyzes the network by propagating a flow's characteristics along its path.

The TFA algorithm for a flow traversing n nodes is as follows [65, Sec. 3.2]:

1. **Initialization:** The flow enters Node 1 with its known fresh arrival curve, α_0.
2. **At Node k (for $k = 1$ to n):**
   - The input to Node k is α^*_{k-1} (with $\alpha^*_0 = \alpha_0$).
   - The per-node delay is computed: $d_k = h(\alpha^*_{k-1}, \beta_k)$.
   - The output arrival curve is computed using Theorem A.3: $\alpha^*_k = \alpha^*_{k-1} \oslash \beta_k$.
3. **Final Bound:** The total end-to-end delay is calculated as the *sum* of the per-node delays:
$$d_{TFA} = \sum_{k=1}^{n} d_k = \sum_{k=1}^{n} h(\alpha^*_{k-1}, \beta_k)$$

This method is modular because the analysis of each node is self-contained, requiring only the incoming arrival curve and the local service curve. This is the approach used in analysis tools like the DISCO Network Calculator [65]. While tractable, this method is known to produce less tight (i.e., more pessimistic) bounds than the end-to-end concatenation theorem because the deconvolution at each step re-compounds the burstiness. To improve on this Mifdaoui & Leydier [66] propose TFA++ which incorporates additional information about cross-traffic to tighten the bounds. Furthermore Thomas et al. [57] extend this approach to enable analysis on cyclic network topologies.

This extended, named FP-TFA++, approach can be broken into two main parts:

1. **The Fixed-Point Iteration:** The high-level process that manages the cyclic dependencies [57].
2. **TFA++ Analysis:** The core TFA++-based calculation performed at each node within one iteration [57], [66].

[1] $f_{\uparrow}(t) = max\{0, sup_{0 \leq s \leq t} f(s)\}$

Part 1: The Fixed-Point Iteration This is the outer loop of the algorithm, designed to find a stable solution for the entire network.

1. **Perform *Cuts*:** The algorithm first identifies all cyclic dependencies. It then virtually *cuts* the network paths at strategic points to render the topology acyclic (i.e., feed-forward).
2. **Initialize Bursts:** It starts with an initial vector of bursts for all flows at the *cut* points. A common starting point is a vector of zeros, $b_{in} = \vec{0}$, representing an empty network.
3. **Run Feed-Forward Analysis (TFA++ Core):** Using the current b_{in} as the known input bursts at the cuts, the algorithm performs a full feed-forward analysis (detailed in Part 2) on the entire acyclic graph.
4. **Compute Output Bursts:** This analysis computes the delay at each node and, crucially, the new output bursts for all flows at the *end* of the virtual cuts. This results in a new vector, b_{out}.
5. **Check for Fixed Point:** The algorithm compares the new burst vector with the one used for the iteration:
   - If $b_{out} = b_{in}$, a **stable fixed point** has been reached. The burst and delay values are stable and valid. The iteration stops.
   - If $b_{out} \neq b_{in}$, the algorithm sets $b_{in} = b_{out}$ and **returns to step 3** for another iteration.
6. **Calculate Final e2e Delay:** Once the fixed point is reached, the end-to-end (e2e) delay for any flow f is the sum of the delays (d_i^f) computed at each node (k) along its path, plus propagation delays (d_{prop}) and source delays (d_{src}^f):

$$d^f = \max_{\forall f_i} \left(d_{src}^f + d_{prop} + \sum_{\forall k \in \phi^{f_i} \setminus src} d_k^f \right)$$

   Where ϕ^{f_i} represents the path of flow f_i, and the analysis may be done for every flow in the network.

Part 2: TFA++ Analysis This part details the TFA++ algorithm [66]. The goal is to compute the delay bounds for all flows traversing the network using a feed-forward analysis that improves upon standard TFA by accounting for the finite transmission capacities of network links. The process involves the following steps for a node n:

1. **Compute the Aggregate Arrival Curve:** Unlike standard TFA, which simply sums the arrival curves of all incoming flows, TFA++ refines this by considering the physical capacity of the incoming links. The aggregate traffic from a predecessor node l cannot

exceed that link's capacity C_l. Let $Pred(n)$ be the set of predecessor nodes to n, and $\gamma_l(t) = C_l \cdot t$ be the capacity curve of link $l \to n$. The aggregate arrival curve α_{agg} at node n is [66, Eq. 3]:

$$\alpha_{agg}(t) = \sum_{l \in Pred(n)} \left(\left(\sum_{f \in flows(l)} \alpha_n^f \right) \otimes \gamma_l \right)(t)$$

This intersection (min-plus convolution with a linear curve) tightens the input model by capping bursts at the link speed.

2. **Compute Delay Bound:** The delay bound d_n is computed using the tightened aggregate arrival curve α_{agg} and the total service curve β_n guaranteed by the node (typically a rate-latency curve $\beta_{R,T}$):
$$d_n = h(\alpha_{agg}, \beta_n)$$

3. **Compute Output Arrival Curves (Serialization Effect):** To propagate the flow to the next node, TFA++ calculates a specific output arrival curve for each flow f. This step incorporates the *serialization effect* of the output link and the residual service available to the flow.

   First, the residual service curve β_n^f for the specific flow f is computed (e.g., using Theorem A.5 for NP-SP). Then, the output arrival curve $\alpha_n^{f,*}$ is derived by convolving the input with both the output link capacity curve $\gamma_n(t) = C_n \cdot t$ and the residual service curve [66, Eq. 5]:
$$\alpha_n^{f,*}(t) = \left((\alpha_n^f \otimes \gamma_n) \oslash \beta_n^f \right)(t) \tag{A.1}$$

   Here, $(\alpha_n^f \otimes \gamma_n)$ represents the traffic shaped by the physical link bandwidth (serialization), and the further convolution with β_n^f represents the service constraint. This results in a tighter bound than the standard deconvolution approach [66, Eq. 6].

A State of The Art tool that applies this method is the WoPANets tool [100], which is used throughout this document whenever using the FP-TFA++ method for delay bound computation.

Appendix B

Implementation of the Partial CBS Deployment Framework

Contents

B.1 Framework Architecture . . . 131
B.1.1 The Network Object Model . . . 132
B.2 Solver Class Structure . . . 132
B.2.1 The Solver Wrapper Class . . . 134
B.2.2 Optimization Logic and Inheritance . . . 134
B.3 Verification Client . . . 134
B.4 Usage Guide . . . 135
B.4.1 Single Instance Execution . . . 136
B.4.2 Automated Batch Processing . . . 136

This appendix details the software implementation of the frameworks presented in Chapters 3, 4, and 5. The solution is developed in Python and is architected as a modular library, the `CBSDeploymentFramework`. It utilizes Object-Oriented Programming (OOP) principles to abstract network elements and implements the optimization logic through a structured class hierarchy. Verification is performed via a TCP socket client interfacing with a version of the *WoPANETS* [100] tool named *Timaeus*.

B.1 Framework Architecture

The software architecture is divided into three distinct layers, ensuring separation of concerns between data modeling, algorithmic logic, and external verification. Figure B.1 illustrates the high-level system architecture and the boundaries between the Python framework and the external verification tool.

1. **Data Modeling Layer:** Responsible for parsing XML network topologies and instantiating them as Python objects.
2. **Optimization Layer:** Contains the classes that implement the deployment algorithms (heuristics and exact methods).

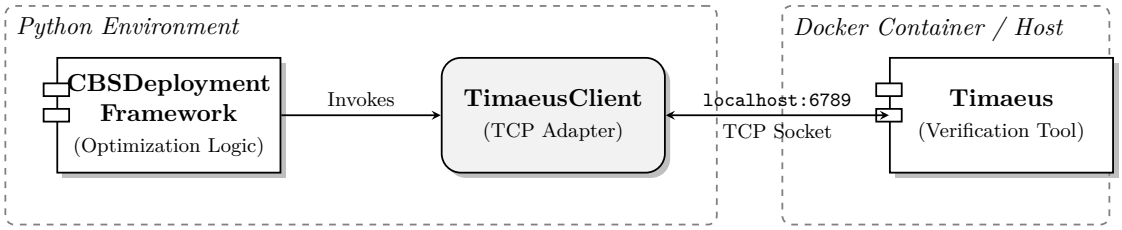


Figure B.1: Component Diagram of the System Architecture. The `CBSDeploymentFramework` interacts with the *Timaeus* verification tool via a TCP socket interface provided by the `TimaeusClient`.

3. **Verification Layer:** Manages network I/O operations to communicate with the external analysis tool.

B.1.1 The Network Object Model

The physical network topology is mapped to a set of Python classes defined within the `CBSDeploymentFramework` package. The relationships between these entities are depicted in the UML class diagram in Figure B.2.

- **`Network` Class (`network.py`):** The primary container class that aggregates lists of `Device`, `OutputPort`, and `Flow` objects. It encapsulates a toolbox to compute network-related statistics such as the flows traversing a specific port, computing which devices have a FRER elimination point, etc.
- **`Device` Class Hierarchy (`device.py`):** Represents network nodes. The base class `Device` is extended by two subclasses: `Switch` (representing forwarding nodes) and `EndSystem` (representing traffic sources and sinks).
- **`OutputPort` Class (`output_port.py`):** Represents a directed edge in the network graph. It stores the configuration state of the priority queues, specifically the `idle_slope` value and the `cbs` boolean flag for each priority level.
- **`Flow` Class (`flow.py`):** Represents a flow traversing the network. It encapsulates attributes such as `rate`, `burst`, and `deadline`. It also maintains the computed `schedulability` state.

B.2 Solver Class Structure

The framework implements multiple optimization algorithms. This is achieved through a specific class structure in `optimization.py`, which uses composition and inheritance to manage different deployment strategies. This design pattern is visualized in Figure B.3.

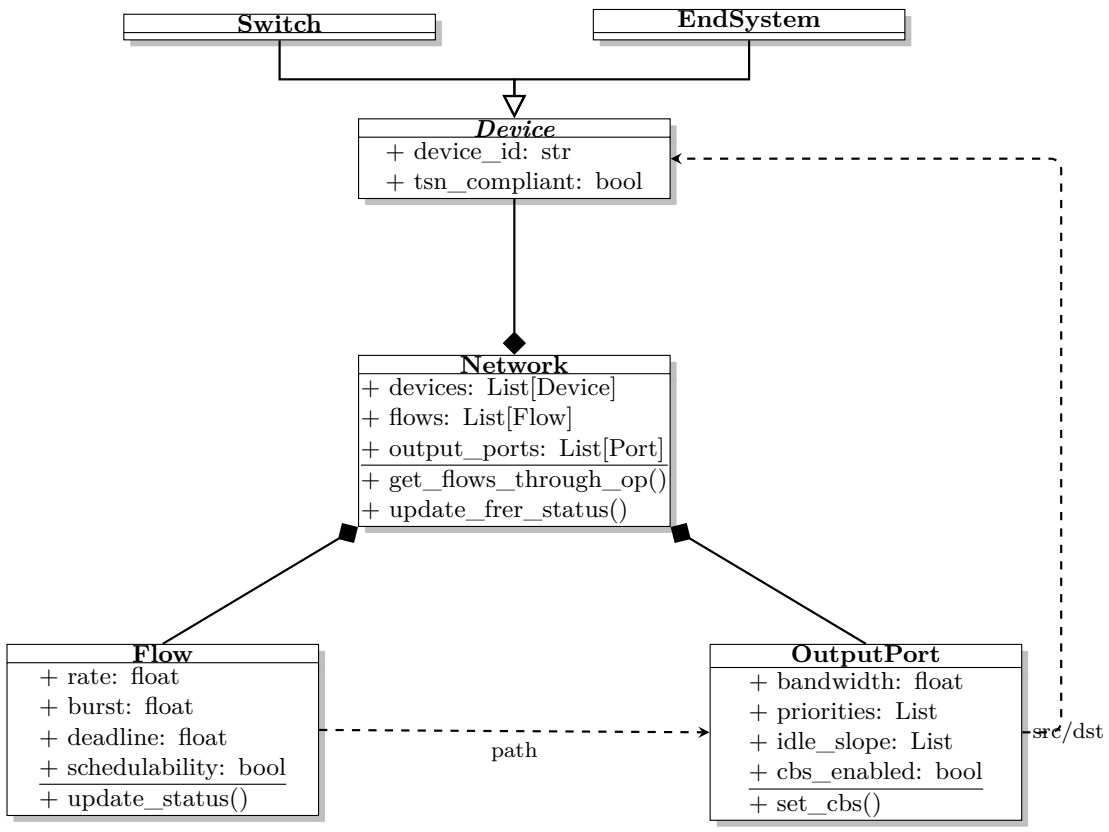


Figure B.2: UML Class Diagram of the Network Object Model. The `Network` class aggregates `Device`, `OutputPort`, and `Flow` objects, with specific inheritance for Switches and EndSystems.

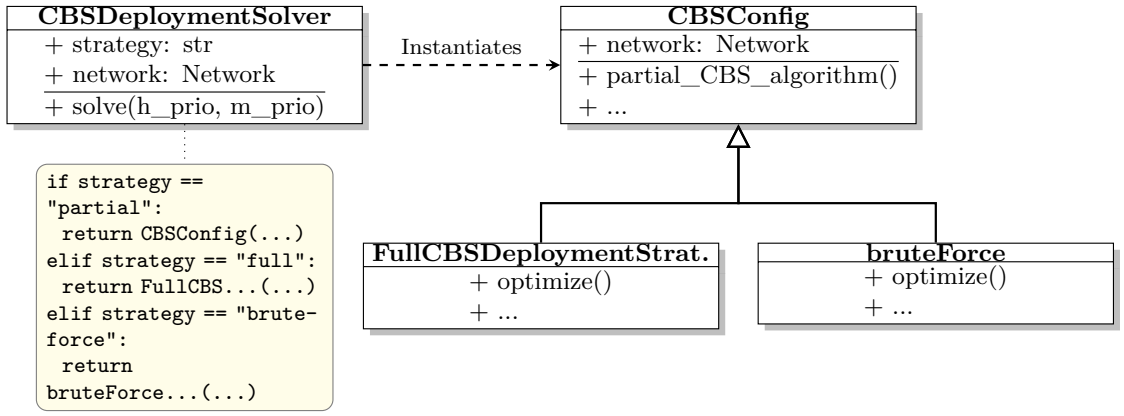


Figure B.3: UML Class Diagram illustrating the Strategy Pattern used for solver selection. The `CBSDeploymentSolver` context delegates the optimization logic to concrete implementations of `CBSConfig`.

B.2.1 The Solver Wrapper Class

The `CBSDeploymentSolver` class acts as the entry point for the optimization process. It abstracts the instantiation complexity of the specific underlying algorithms.

```
class CBSDeploymentSolver:
    def __init__(self, network, strategy="partial", ...):
        self.strategy = strategy
        # Configuration parameters...
```

Listing B.1: Initialization of the Solver Class

The `solve()` method of this class inspects the `strategy` argument and instantiates the corresponding optimization class:

- `strategy="partial"`: Instantiates the `CBSConfig` class. This executes the heuristic algorithms for Partial CBS deployment, including the logic for FRER-awareness.
- `strategy="full"`: Instantiates the `FullCBSDeploymentStrategy` class.
- `strategy="brute-force"`: Instantiates the `bruteForce` class.

B.2.2 Optimization Logic and Inheritance

The core optimization logic is implemented via an inheritance hierarchy:

- **`CBSConfig` (Base Class):** This class implements the fundamental methods required for the Partial CBS algorithm. It defines the `select_device` method to compute the device based on a flow of interest, the `compute_min_idleslope` method for calculating bandwidth reservation, and the `reconfiguration` method for iterative parameter adjustment.
- **`FullCBSDeploymentStrategy` (Subclass):** Inherits from `CBSConfig`. It overrides specific behaviors to implement the IdleSlope search algorithm, which optimizes configurations under the constraint that CBS is deployed on all switches.
- **`BruteForce` (Subclass):** Inherits from `CBSConfig`. This class formulates the placement problem as an ILP. It utilizes the Gurobi optimizer to provide feasible combinations under the current constraints in an optimized fashion. It then computes potential CBS deployments under the provided potential solution until finding a combination that is validated by the external verification tool.

B.3 Verification Client

Verification of the solution is delegated to the *Timaeus* tool. The communication interface is implemented in the `TimaeusClient` class.

- **Socket Abstraction:** The class inherits from a `TCPSocket` base class, managing the low-level TCP connection to the localhost port 6789.

- **Remote Procedure Calls:** The client implements methods to serialize commands (e.g., `timaeus_set_request`) and deserialize responses. This allows the Python framework to modify the Timaeus model state (e.g., setting `IdleSlope`) and retrieve computation results (e.g., end-to-end delay bounds).

- **Data Retrieval:** Aside from end-to-end delay bounds, to support FRER analysis, the framework specifically requests the `delaysPerPort` collection from Timaeus, allowing the `Network` class to reconstruct cumulative delays across redundant paths. It also retrieves flow-level statistics to correctly determine IdleSlope values for elimination points.

B.4 Usage Guide

The framework is designed to be executed either as a standalone analysis script for a single topology or as part of an automated pipeline for batch processing benchmarks. The interaction sequence between the user script, the solver strategies, and the verification client is depicted in Figure B.4.

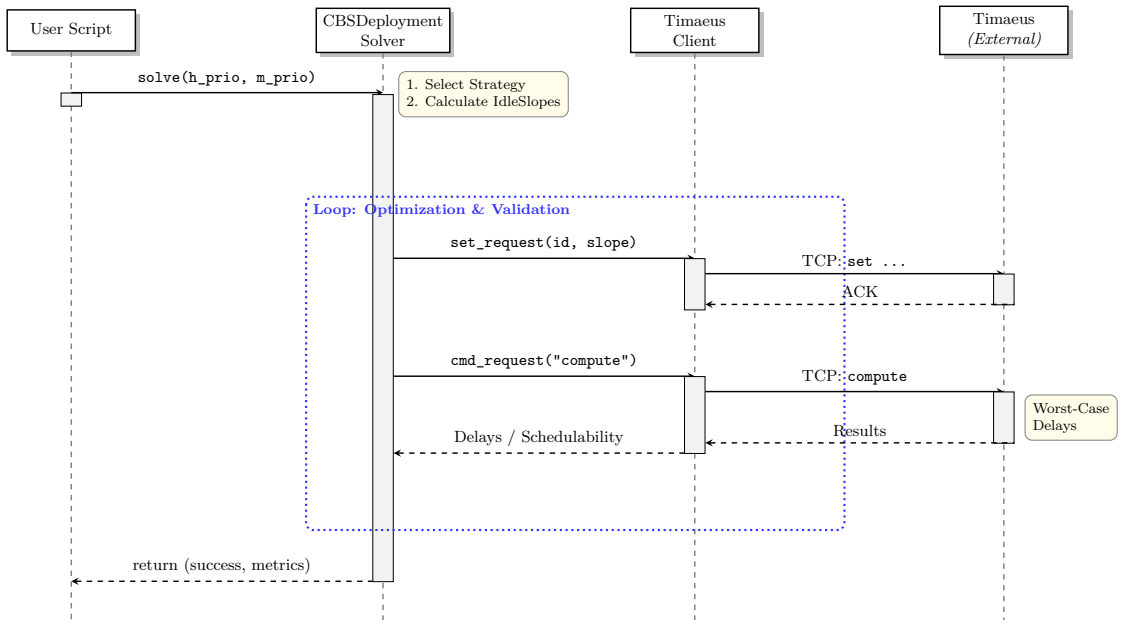


Figure B.4: Sequence Diagram showing the optimization workflow. The user script initiates the solver, which iteratively configures the network parameters and validates them via the `TimaeusClient`.

B.4.1 Single Instance Execution

The following script shows a possible entry point for single instance execution. The script illustrates the complete workflow: creating the object model, instantiating the optimization object, executing the solver, and exporting performance metrics.

```
import os
from CBSDeploymentFramework import (
    Timaeus_XMLParser,
    CBSDeploymentSolver,
    NetworkExporter
)
from TimaeusClient.TimaeusClient import TimaeusClient

def run_analysis(xml_path):
    # 1. Parse the input network topology
    parser = Timaeus_XMLParser(xml_path)
    network = parser.parsed_network

    # 2. Instantiate the Solver with the Partial Strategy
    solver = CBSDeploymentSolver(
        network,
        strategy="partial",
        FRER_active=False
    )

    # 3. Execute Optimization
    # Returns success status and runtime metrics
    success, t_framework, t_core = solver.solve(h_prio=0, m_prio=1)

    if success:
        print(f"Solution found in {t_framework:.3f}s (Core algo: {t_core:.3f}s)
    ")

        # 4. Export Results for Analysis
        exporter = NetworkExporter(network)
        results_dir = "Results"
        os.makedirs(results_dir, exist_ok=True)
        exporter.export_cbs_info(os.path.join(results_dir, "cbs_config.csv"))
        exporter.export_flow_info(os.path.join(results_dir, "flow_stats.csv"))
    else:
        print("Optimization failed to find a schedulable configuration.")

if __name__ == "__main__":
    run_analysis("benchmarks/sample_topology.xml")
```

Listing B.2: Single Instance Execution Script

B.4.2 Automated Batch Processing

For large-scale evaluation, the framework includes an automation script. This script manages the lifecycle of the external verification tool (*Timaeus*) via Docker containers and orchestrates

the execution of the analysis script across a hierarchy of test scenarios.

```
import subprocess
import time
from pathlib import Path

# Configuration constants
DOCKER_IMAGE = "timaeus"
SERVER_STARTUP_WAIT = 10  # seconds
BASE_DIR = Path("/home/user/CBS_Framework")
ANALYSIS_SCRIPT = BASE_DIR / "\ac{CBS} Deployment/main.py"

def run_single_test_instance(tmne_file_path, results_dir):
    """
    Orchestrates a single test: starts the Timaeus container,
    runs the solver, and cleans up.
    """
    container_name = f"timaeus-test-{tmne_file_path.stem}"

    # 1. Start Timaeus Server in Docker
    docker_cmd = [
        "docker", "run", "--rm", "--net=host",
        "--name", container_name,
        "-v", f"{BASE_DIR}:/mnt/data",
        DOCKER_IMAGE,
        "/root/Timaeus/Linux/Timaeus", "-server"
    ]

    try:
        print(f"Starting verification server for {tmne_file_path.name}...")
        subprocess.Popen(docker_cmd)
        time.sleep(SERVER_STARTUP_WAIT) # Wait for server initialization

        # 2. Run the Python Analysis Client (Subprocess)
        xml_file = str(tmne_file_path).replace(".tmne", ".xml")
        log_file = results_dir / "results.json"

        client_cmd = [
            "python3", str(ANALYSIS_SCRIPT),
            xml_file,
            "--log_file", str(log_file)
        ]

        subprocess.run(client_cmd, check=True, timeout=600)

    except subprocess.TimeoutExpired:
        print(f"Test timed out for {tmne_file_path.name}")
    finally:
        # 3. Teardown: Stop the Docker Container
        subprocess.run(["docker", "stop", container_name], check=False)

if __name__ == "__main__":
    # Example directory traversal
    test_dir = BASE_DIR / "datasets/Synthetic"
    for test_file in test_dir.glob("*.tmne"):
```

```
54          run_single_test_instance(test_file, Path("Results/"))
```

Listing B.3: Automated Batch Processing Script

Bibliography

[1] L. L. Bello, "Novel trends in automotive networks: A perspective on ethernet and the ieee audio video bridging," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, 2014, pp. 1–8 (cit. on p. 7).

[2] L. Deng, G. Xie, H. Liu, Y. Han, R. Li, and K. Li, "A survey of real-time ethernet modeling and design methodologies: From avb to tsn," *ACM Comput. Surv.*, vol. 55, no. 2, Jan. 2022 (cit. on pp. 7, 8).

[3] L. Lo Bello, E. Bini, and G. Patti, "Priority-driven swapping-based scheduling of aperiodic real-time messages over ethercat networks," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 3, pp. 741–751, 2015 (cit. on p. 7).

[4] G. S. Sestito et al., "A method for anomalies detection in real-time ethernet data traffic applied to profinet," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2171–2180, 2018 (cit. on p. 7).

[5] T. Steinbach, F. Korf, and T. C. Schmidt, "Real-time ethernet for automotive applications: A solution for future in-car networks," in *2011 IEEE International Conference on Consumer Electronics -Berlin (ICCE-Berlin)*, 2011, pp. 216–220 (cit. on p. 7).

[6] K. Erwinski, M. Paprocki, L. M. Grzesiak, K. Karwowski, and A. Wawrzak, "Application of ethernet powerlink for communication in a linux rtai open cnc system," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 2, pp. 628–636, 2013 (cit. on p. 7).

[7] *Avnu Alliance | About Us*, en-US, `https://avnu.org/about-us/` (cit. on p. 8).

[8] AVNU Alliance, "Avb for professional a/v use," 2009 (cit. on p. 8).

[9] *Time-Sensitive Networking (TSN) Task Group*, `https://1.ieee802.org/tsn/` (cit. on pp. 8, 11).

[10] O. A. Hotescu, "Vers la convergence de réseaux dans l'avionique," Theses, Institut National Polytechnique de Toulouse - INPT, May 2020 (cit. on p. 8).

[11] S. International, *664p7-1 aircraft data network, part 7, avionics full-duplex switched ethernet network*, https://aviation-ia.sae-itc.com/standards/arinc664p7-1-664p7-1-aircraft-data-network-part-7-avionics-full-duplex-switched-ethernet-network (cit. on p. 8).

[12] A. FINZI, A. MIFDAOUI, F. FRANCES, and E. LOCHIN, "Network calculus-based timing analysis of afdx networks with strict priority and tsn/bls shapers," in *2018 IEEE 13th International Symposium on Industrial Embedded Systems (SIES)*, 2018, pp. 1–10 (cit. on p. 8).

[13] J.-D. Decotignie, "Ethernet-based real-time and industrial communications," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1102–1117, 2005 (cit. on p. 8).

[14] J. Diemer, J. Rox, R. Ernst, F. Chen, K.-T. Kremer, and K. Richter, "Exploring the worst-case timing of ethernet avb for industrial applications," in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, 2012, pp. 3182–3187 (cit. on p. 8).

[15] S. Kramer, D. Zuegenbein, and A. Hamann, "Real world automotive benchmarks for free," pp. 1–6, 2015, `https://rtn.ecrts.org/forum/download/WATERS15_Real_World_Automotive_Benchmark_For_Free.pdf` (cit. on p. 8).

[16] A. Finzi, *Specification and analysis of an extended afdx with tsn/bls shapers for mixed-criticality avionics applications*, eng, Institut supérieur de l'aéronautique et de l'espace Toulouse, 2018 (cit. on pp. 8, 9).

[17] P.-J. Chaine, *Adaptabilité de Time Sensitive Networking aux exigences de l'industrie aérospatiale*, eng, 2022 (cit. on pp. 8, 9, 11, 16).

[18] Q. Bailleul, "Dimensioning TSN network synchronization in different embedded contexts," Theses, Institut National Polytechnique de Toulouse - INPT, Nov. 2023 (cit. on pp. 8, 9, 14).

[19] M. Ashjaei, L. Lo Bello, M. Daneshtalab, G. Patti, S. Saponara, and S. Mubeen, "Time-Sensitive Networking in automotive embedded systems: State of the art and research opportunities," *Journal of Systems Architecture*, vol. 117, p. 102 137, Aug. 2021 (cit. on pp. 8, 14, 16).

[20] E. Grossman, *Deterministic Networking Use Cases*, RFC 8578, May 2019 (cit. on p. 8).

[21] L. Thiele and R. Wilhelm, "Design for timing predictability," *Real-Time Systems*, vol. 28, no. 2, pp. 157–177, 2004 (cit. on p. 8).

[22] E. Tovar and F. Vasques, "Real-time fieldbus communications using profibus networks," *IEEE Transactions on Industrial Electronics*, vol. 46, no. 6, pp. 1241–1251, 1999 (cit. on p. 9).

[23] J.-L. Scharbarg, M. Boyer, and C. Fraboul, "Can-ethernet architectures for real-time applications," in *2005 IEEE Conference on Emerging Technologies and Factory Automation*, vol. 2, 2005, 8 pp.–252 (cit. on p. 9).

[24] S. Gopalakrishnan, L. Sha, and M. Caccamo, "Hard real-time communication in bus-based networks," in *25th IEEE International Real-Time Systems Symposium*, 2004, pp. 405–414 (cit. on p. 9).

[25] "Ieee standard for information technology-telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements part 3: Carrier sense multiple access with collision detection (csma/cd) access method and physical layer specifications," *IEEE Std 802.3-2002 (Revision of IEEE Std 802.3, 2000 edn)*, pp. 1–1550, 2002 (cit. on p. 9).

[26] "Ieee standard for local and metropolitan area networks—virtual bridged local area networks," *IEEE Std 802.1Q-2005 (Incorporates IEEE Std 802.1Q1998, IEEE Std 802.1u-2001, IEEE Std 802.1v-2001, and IEEE Std 802.1s-2002)*, pp. 1–300, 2006 (cit. on pp. 9, 10).

[27] L. Lo Bello, G. Patti, and L. Leonardi, "A perspective on ethernet in automotive communicationscurrent status and future trends," *Applied Sciences*, vol. 13, no. 3, 2023 (cit. on p. 9).

[28] A. Berisa, M. Ashjaei, M. Daneshtalab, M. Sjödin, and S. Mubeen, "Investigating and analyzing can-to-tsn gateway forwarding techniques," in *2023 IEEE 26th International Symposium on Real-Time Distributed Computing (ISORC)*, 2023, pp. 136–145 (cit. on pp. 10, 14).

[29] "Ieee standard for local and metropolitan area networks–bridges and bridged networks," *IEEE Std 802.1Q-2022 (Revision of IEEE Std 802.1Q-2018)*, pp. 1–2163, 2022 (cit. on pp. 10, 12, 13, 22, 23).

[30] "Ieee standard for local and metropolitan area networks–frame replication and elimination for reliability," *IEEE Std 802.1CB-2017*, pp. 1–102, 2017 (cit. on pp. 10, 11).

[31] "Ieee standard for local and metropolitan area networks–bridges and bridged networks–amendment 28: Per-stream filtering and policing," *IEEE Std 802.1Qci-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, and IEEE Std 802.1Qbz-2016)*, pp. 1–65, 2017 (cit. on pp. 10, 11).

[32] "IEEE Standard for Local and metropolitan area networks– Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams," *IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005)*, pp. 1–72, 2010 (cit. on pp. 10, 13, 15, 17, 22, 37, 38).

[33] "Ieee standard for local and metropolitan area networks–bridges and bridged networks - amendment 34: Asynchronous traffic shaping," *IEEE Std 802.1Qcr-2020 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018, IEEE Std 802.1Qcc-2018, IEEE Std 802.1Qcy-2019, and IEEE Std 802.1Qcx-2020)*, pp. 1–151, 2020 (cit. on p. 12).

[34] "Ieee standard for local and metropolitan area networks – bridges and bridged networks - amendment 25: Enhancements for scheduled traffic," *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)*, pp. 1–57, 2016 (cit. on pp. 12, 23).

[35] "Ieee standard for local and metropolitan area networks–bridges and bridged networks–amendment 29: Cyclic queuing and forwarding," *IEEE 802.1Qch-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd(TM)-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, IEEE Std 802.1Qbz-2016, and IEEE Std 802.1Qci-2017)*, pp. 1–30, 2017 (cit. on p. 12).

[36] "Ieee standard for local and metropolitan area networks – bridges and bridged networks – amendment 26: Frame preemption," *IEEE Std 802.1Qbu-2016 (Amendment to IEEE Std 802.1Q-2014)*, pp. 1–52, 2016 (cit. on p. 13).

[37] M. Björklund, *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)*, RFC 6020, Oct. 2010 (cit. on p. 14).

[38] J. H. Kim, S.-H. Seo, N.-T. Hai, B. M. Cheon, Y. S. Lee, and J. W. Jeon, "Gateway framework for in-vehicle networks based on can, flexray, and ethernet," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 10, pp. 4472–4486, 2015 (cit. on p. 14).

[39] "Iso/iec/ieee–information technology–telecommunications and information exchange between systems–local and metropolitan area networks–part 1as: Timing and synchronization for time-sensitive applications in bridged local area networks technical corrigendum 1: Technical and editorial corrections," *ISO/IEC/IEEE 8802-1AS:2021/Cor.1:2023(E)*, pp. 1–33, 2023 (cit. on p. 14).

[40] M. L. Raagaard and P. Pop, "Optimization algorithms for the scheduling of ieee 802.1 time-sensitive networking (tsn)," *Tech. Univ. Denmark, Lyngby, Denmark, Tech. Rep*, vol. 1, 2017 (cit. on p. 14).

[41] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in ieee 802.1qbv time sensitive networks," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS '16, Brest, France: Association for Computing Machinery, 2016, pp. 183192 (cit. on pp. 14, 16).

[42] S. S. Craciunas, R. S. Oliver, and T Ag, "An overview of scheduling mechanisms for time-sensitive networks," *Proceedings of the Real-time summer school LÉcole dÉté Temps Réel (ETR)*, pp. 1551–3203, 2017 (cit. on p. 14).

[43] R. Serna Oliver, S. S. Craciunas, and W. Steiner, "Ieee 802.1qbv gate control list synthesis using array theory encoding," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2018, pp. 13–24 (cit. on p. 14).

[44] F. Dürr and N. G. Nayak, "No-wait packet scheduling for ieee time-sensitive networks (tsn)," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS '16, Brest, France: Association for Computing Machinery, 2016, pp. 203212 (cit. on pp. 14, 16).

[45] M. Pahlevan and R. Obermaisser, "Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2018, pp. 337–344 (cit. on p. 14).

[46] V. Gavriluţ and P. Pop, "Traffic-type assignment for tsn-based mixed-criticality cyber-physical systems," *ACM Transactions on Cyber-Physical Systems*, vol. 4, no. 2, pp. 1–27, 2020 (cit. on p. 14).

[47] J. Specht and S. Samii, "Synthesis of queue and priority assignment for asynchronous traffic shaping in switched ethernet," in *2017 IEEE Real-Time Systems Symposium (RTSS)*, 2017, pp. 178–187 (cit. on p. 15).

[48] J. Diemer, D. Thiele, and R. Ernst, "Formal worst-case timing analysis of Ethernet topologies with strict-priority and AVB switching," in *7th IEEE International Symposium on Industrial Embedded Systems (SIES'12)*, ISSN: 2150-3117, Jun. 2012, pp. 1–10 (cit. on pp. 15, 21, 22, 24).

[49] M. Ashjaei, G. Patti, M. Behnam, T. Nolte, G. Alderisi, and L. LoăBello, "Schedulability analysis of Ethernet Audio Video Bridging networks with scheduled traffic support," en, *Real-Time Systems*, vol. 53, no. 4, pp. 526–577, Jul. 2017 (cit. on pp. 15, 20, 22–24, 92).

[50] J. Cao, M. Ashjaei, P. J. L. Cuijpers, R. J. Bril, and J. J. Lukkien, "An independent yet efficient analysis of bandwidth reservation for credit-based shaping," in *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, Jun. 2018, pp. 1–10 (cit. on p. 15).

[51] J.-Y. Le Boudec and P. Thiran, *Network Calculus* (Lecture Notes in Computer Science). Springer, 2001 (cit. on pp. 15, 21, 38, 121–126).

[52] L. Maile, K.-S. J. Hielscher, and R. German, "Delay-Guaranteeing Admission Control for Time-Sensitive Networking Using the Credit-Based Shaper," *IEEE Open Journal of the Communications Society*, vol. 3, pp. 1834–1852, 2022 (cit. on pp. 15, 16, 22, 23, 92).

[53] L. Zhao, Y. Yan, and X. Zhou, "Minimum Bandwidth Reservation for CBS in TSN With Real-Time QoS Guarantees," *IEEE Transactions on Industrial Informatics*, vol. 20, no. 4, pp. 6187–6198, Apr. 2024 (cit. on pp. 15, 16, 22, 23).

[54] S. Yu, F. He, A. Xie, and L. Zhao, "Efficient adaptive bandwidth allocation for deadline-aware online admission control in centralized time-sensitive networking," *Journal of Systems Architecture*, vol. 168, p. 103 539, 2025 (cit. on pp. 15, 22, 23).

[55] V. Gavriluţ and P. Pop, "Traffic-type assignment for tsn-based mixed-criticality cyber-physical systems," *ACM Trans. Cyber-Phys. Syst.*, vol. 4, no. 2, Jan. 2020 (cit. on p. 16).

[56] P.-J. Chaine, M. Boyer, C. Pagetti, and F. Wartel, "Egress-tt configurations for tsn networks," in *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, ser. RTNS '22, Paris, France: Association for Computing Machinery, 2022, pp. 5869 (cit. on p. 16).

[57] L. Thomas, J.-Y. Le Boudec, and A. Mifdaoui, "On cyclic dependencies and regulators in time-sensitive networks," in *2019 IEEE Real-Time Systems Symposium (RTSS)*, 2019, pp. 299–311 (cit. on pp. 16, 127).

[58] L. Thomas and J.-Y. Le Boudec, "On time synchronization issues in time-sensitive networks with regulators and nonideal clocks," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, no. 2, Jun. 2020 (cit. on p. 17).

[59] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and Microprogramming*, Parallel Processing in Embedded Real-time Systems, vol. 40, no. 2, pp. 117–134, Apr. 1994 (cit. on p. 20).

[60] Tindell, Hansson, and Wellings, "Analysing real-time communications: Controller area network (can)," in *1994 Proceedings Real-Time Systems Symposium*, 1994, pp. 259–263 (cit. on p. 20).

[61] F. Bonomi, S. Montagna, and R. Paglino, "Busy period analysis for an atm switching element output line," in *[Proceedings] IEEE INFOCOM '92: The Conference on Computer Communications*, 1992, 544–551 vol.2 (cit. on p. 20).

[62] U. D. Bordoloi, A. Aminifar, P. Eles, and Z. Peng, "Schedulability analysis of Ethernet AVB switches," in *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, ISSN: 2325-1301, Aug. 2014, pp. 1–10 (cit. on p. 20).

[63] J. Cao, P. J. Cuijpers, R. J. Bril, and J. J. Lukkien, "Independent yet Tight WCRT Analysis for Individual Priority Classes in Ethernet AVB," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS '16, New York, NY, USA: Association for Computing Machinery, Oct. 2016, pp. 55–64 (cit. on p. 20).

[64] D. Bujosa, J. Proenza, A. V. Papadopoulos, T. Nolte, and M. Ashjaei, "An Improved Worst-Case Response Time Analysis for AVB Traffic in Time-Sensitive Networks," in *2024 IEEE Real-Time Systems Symposium (RTSS)*, ISSN: 2576-3172, Dec. 2024, pp. 135–147 (cit. on pp. 20, 22).

[65] J. B. Schmitt and F. A. Zdarsky, "The disco network calculator: A toolbox for worst case analysis," in *Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools*, ser. valuetools '06, Pisa, Italy: Association for Computing Machinery, 2006, 8es (cit. on pp. 20, 127).

[66] A. Mifdaoui and T. Leydier, "Beyond the Accuracy-Complexity Tradeoffs of CompositionalAnalyses using Network Calculus for Complex Networks," in *Proceedings of the 10th International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (co-located with RTSS 2017)*, Paris, France, Dec. 2017, pp. 1–8 (cit. on pp. 20, 22, 45, 127–129).

[67] "Modular analysis: Computing with curves," in *Deterministic Network Calculus*. John Wiley & Sons, Ltd, 2018, ch. 10, pp. 229–255. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119440284.ch10` (cit. on pp. 20, 22).

[68] S. Martin, "Maîtrise de la dimension temporelle de la qualité de service dans les réseaux," Theses, Université Paris XII Val de Marne, Jul. 2004 (cit. on p. 20).

[69] X. Li and L. George, "Deterministic delay analysis of AVB switched Ethernet networks using an extended Trajectory Approach," en, *Real-Time Systems*, vol. 53, no. 1, pp. 121–186, Jan. 2017 (cit. on p. 21).

[70] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis the symta/s approach," *IEE Proceedings - Computers and Digital Techniques*, vol. 152, pp. 148–166, 2 2005. eprint: `https://digital-library.theiet.org/doi/pdf/10.1049/ip-cdt%3A20045088` (cit. on p. 21).

[71] P. Axer, D. Thiele, R. Ernst, and J. Diemer, "Exploiting shaper context to improve performance bounds of Ethernet AVB networks," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, ISSN: 0738-100X, Jun. 2014, pp. 1–6 (cit. on pp. 21, 22).

[72] J.-Y. Le Boudec, "Network calculus made easy," Jan. 1997 (cit. on p. 21).

[73] F. Baccelli, G. Cohen, G. Olsder, and J.-P. Quadrat, "Synchronization and linearity - an algebra for discrete event systems," *The Journal of the Operational Research Society*, vol. 45, Jan. 1994 (cit. on p. 21).

[74] R. Queck, "Analysis of Ethernet AVB for automotive networks using Network Calculus," in *2012 IEEE International Conference on Vehicular Electronics and Safety (ICVES 2012)*, Jul. 2012, pp. 61–67 (cit. on p. 21).

[75] J. A. R. De Azua and M. Boyer, "Complete modelling of AVB in Network Calculus Framework," in *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, ser. RTNS '14, New York, NY, USA: Association for Computing Machinery, Oct. 2014, pp. 55–64 (cit. on p. 21).

[76] E. Mohammadpour, E. Stai, and J.-Y. Le Boudec, "Improved Credit Bounds for the Credit-Based Shaper in Time-Sensitive Networking," *IEEE Networking Letters*, vol. 1, no. 3, pp. 136–139, Sep. 2019 (cit. on pp. 21, 30).

[77] L. Zhao, P. Pop, Z. Zheng, and Q. Li, "Timing Analysis of AVB Traffic in TSN Networks Using Network Calculus," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Apr. 2018, pp. 25–36 (cit. on p. 21).

[78] F. Reimann, S. Graf, F. Streit, M. GlaSS, and J. Teich, "Timing analysis of Ethernet AVB-based automotive E/E architectures," in *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, ISSN: 1946-0759, Sep. 2013, pp. 1–8 (cit. on p. 21).

[79] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse, "System architecture evaluation using modular performance analysis: A case study," en, *International Journal on Software Tools for Technology Transfer*, vol. 8, no. 6, pp. 649–667, Oct. 2006 (cit. on p. 21).

[80] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *2000 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 4, 2000, 101–104 vol.4 (cit. on p. 21).

[81] N. Benammar, H. Bauer, F. Ridouard, and P. Richard, "Timing analysis of AVB Ethernet network using the Forward end-to-end Delay Analysis," in *Proceedings of the 26th International Conference on Real-Time Networks and Systems*, ser. RTNS '18, New York, NY, USA: Association for Computing Machinery, Oct. 2018, pp. 223–233 (cit. on p. 21).

[82] J. Cao, P. J. L. Cuijpers, R. J. Bril, and J. J. Lukkien, "Independent WCRT analysis for individual priority classes in Ethernet AVB," en, *Real-Time Systems*, vol. 54, no. 4, pp. 861–911, Oct. 2018 (cit. on p. 22).

[83] W. Kong, M. Nabi, and K. Goossens, "SynAVB: Route and Slope Synthesis Ensuring Guaranteed Service in Ethernet AVB," in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2022, pp. 1–8 (cit. on pp. 22, 23).

[84] B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, and S. Mubeen, "Bandwidth Reservation Analysis for Schedulability of AVB Traffic in TSN," in *2024 IEEE International Conference on Industrial Technology (ICIT)*, ISSN: 2643-2978, Mar. 2024, pp. 1–8 (cit. on pp. 22, 23).

[85] "IEEE Standard for Local and metropolitan area networks–Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol (SRP)," *IEEE Std 802.1Qat-2010 (Revision of IEEE Std 802.1Q-2005)*, pp. 65–66, 2010 (cit. on pp. 22, 38).

[86] M. Nitta, Y. Ito, and A. Yoshimura, "A Proposal of Method of Finding Appropriate idleSlope Values Over In-Vehicle Ethernet With CBS," in *2022 13th International Conference on Information and Communication Technology Convergence (ICTC)*, ISSN: 2162-1241, Oct. 2022, pp. 1406–1408 (cit. on p. 23).

[87] M. Boyer and H. Daigmorte, "Impact on credit freeze before gate closing in cbs and gcl integration into tsn," in *Proceedings of the 27th International Conference on Real-Time Networks and Systems*, ser. RTNS '19, Toulouse, France: Association for Computing Machinery, 2019, pp. 8089 (cit. on pp. 23, 24).

[88] P. Meyer, T. Steinbach, F. Korf, and T. C. Schmidt, "Extending ieee 802.1 avb with time-triggered scheduling: A simulation study of the coexistence of synchronous and asynchronous traffic," in *2013 IEEE Vehicular Networking Conference*, 2013, pp. 47–54 (cit. on p. 24).

[89] L. Zhao, F. He, E. Li, and H. Xiong, "Improving Worst-Case Delay Analysis for Traffic of Additional Stream Reservation Class in Ethernet-AVB Network," en, *Sensors*, vol. 18, no. 11, p. 3849, Nov. 2018, Publisher: Multidisciplinary Digital Publishing Institute (cit. on p. 24).

[90] E. Mohammadpour, E. Stai, M. Mohiuddin, and J.-Y. Le Boudec, "Latency and Backlog Bounds in Time-Sensitive Networking with Credit Based Shapers and Asynchronous Traffic Shaping," in *2018 30th International Teletraffic Congress (ITC 30)*, vol. 02, Sep. 2018, pp. 1–6 (cit. on p. 25).

[91] L. Zhao, P. Pop, and S. Steinhorst, "Quantitative Performance Comparison of Various Traffic Shapers in Time-Sensitive Networking," *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 2899–2928, Sep. 2022, Conference Name: IEEE Transactions on Network and Service Management (cit. on p. 25).

[92] L. Thomas, A. Mifdaoui, and J.-Y. Le Boudec, "Worst-case delay bounds in time-sensitive networks with packet replication and elimination," *IEEE/ACM Transactions on Networking*, vol. 30, no. 6, pp. 2701–2715, 2022 (cit. on pp. 25, 64, 67, 110).

[93] T. Lübeck, P. Meyer, T. Häckel, F. Korf, and T. C. Schmidt, *Asynchronous Traffic Shaping and Redundancy: Avoiding Unbounded Latencies in In-Car Networks*, arXiv:2504.01946 [cs], Apr. 2025 (cit. on p. 25).

[94] L. Zhang, T. Zhang, W. Wu, X. Feng, G. Lin, and F. Ren, "Fault- Tolerant Cyclic Queuing and Forwarding in Time-Sensitive Networking," in *2024 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2024, pp. 1–6 (cit. on p. 25).

[95] L. Zhao, P. Pop, Z. Zheng, H. Daigmorte, and M. Boyer, "Improving Worst-Case End-to-End Delay Analysis of Multiple Classes of AVB Traffic in TSN Networks using Network Calculus," 2019 (cit. on p. 30).

[96] J. Diemer, D. Thiele, and R. Ernst, "Formal worst-case timing analysis of ethernet topologies with strict-priority and avb switching," in *7th IEEE International Symposium on Industrial Embedded Systems (SIES'12)*, 2012, pp. 1–10 (cit. on p. 32).

[97] "IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks," *IEEE Std 802.1Q-2022 (Revision of IEEE Std 802.1Q-2018)*, pp. 219–221, 2022 (cit. on p. 37).

[98] D. Hochbaum, "Complexity and algorithms for nonlinear optimization problems," *Annals of Operations Research*, vol. 153, pp. 257–296, Jun. 2007 (cit. on p. 40).

[99] Gurobi Optimization, LLC, *Tolerances and User-Scaling - Gurobi Optimizer Reference Manual*, n.d. (Cit. on p. 43).

[100] A. Mifdaoui and H. Ayed, "Wopanets: A tool for worst case performance analysis of embedded networks," in *2010 15th IEEE International Workshop on Computer Aided Modeling, Analysis and Design of Communication Links and Networks (CAMAD)*, 2010, pp. 91–95 (cit. on pp. 45, 50, 129, 131).

[101] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd. The MIT Press, 2001 (cit. on p. 54).

[102] P. Cuenot and Q. Bailleul, "Evaluation of Time-Sensitive Networking for future centralized architecture," in *CESA 2025*, SIA, Paris, France, Feb. 2025 (cit. on pp. 55–57).

[103] A. Bouillard, M. Boyer, and E. Corronc, *Deterministic Network Calculus: From Theory to Practical Implementation*. Wiley, 2018 (cit. on p. 126).

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Université de Toulouse

Titre : Méthodologie pour le déploiement de Credit-Based Shaper dans les réseaux sensibles au temps : conception, analyse et optimisation

Mots clés : Time-Sensitive Networking (TSN), Credit-Based Shaper (CBS), Frame-Replication and Elimination for Reliability (FRER), Network Calculus, Interoperability

Résumé : Les systèmes temps réel dans des domaines tels que l'aérospatiale, l'automobile et l'automatisation industrielle migrent de plus en plus vers les réseaux Ethernet Time-Sensitive Networking (TSN) pour répondre à des exigences strictes de temporalité et de fiabilité. Bien que des mécanismes comme le Credit-Based Shaper (CBS) offrent un équilibre entre déterminisme et complexité, la littérature existante suppose majoritairement un « déploiement complet du CBS », où chaque commutateur du réseau est compatible TSN. Cette hypothèse néglige la réalité économique et pratique des réseaux hétérogènes qui doivent intégrer des équipements Ethernet traditionnels. De plus, l'interaction entre le CBS et les mécanismes de fiabilité, spécifiquement la réplication et l'élimination de trames pour la fiabilité (FRER), reste peu étudiée, malgré le fait avéré que le FRER augmente considérablement les rafales de trafic, problème spécifiquement traité par le CBS.
Cette thèse propose une nouvelle méthodologie pour le « déploiement partiel du CBS », dans les réseaux sensibles au temps. Nous formulons la stratégie de déploiement comme un problème d'optimisation visant à minimiser le nombre d'équipements compatibles TSN tout en garantissant les exigences d'ordonnançabilité au pire cas à l'aide du Calcul Réseau (Network Calculus). Nous introduisons une approche heuristique qui place et configure stratégiquement les instances de CBS pour atténuer les rafales de trafic uniquement là où cela est nécessaire, assurant ainsi l'interopérabilité avec l'infrastructure existante. Le cadre proposé est validé au travers d'un cas d'usage industriel représentatif, où nous avons observé le respect des contraintes d'ordonnançabilité couplé à une réduction des coûts.
Cette méthodologie est ensuite étendue pour prendre en compte le FRER (FRER-aware), traitant l'augmentation des rafales introduite par ce mécanisme. Cette approche étendue neutralise les effets de la redondance sur la latence du réseau, afin de minimiser l'utilisation d'équipements compatibles TSN. Cette extension est validée par le même cas d'usage industriel représentatif utilisé pour la version originelle de la méthodologie. Ici, les observations sont cohérentes avec celles relevées pour l'approche originelle.
Enfin, la thèse évalue l'optimalité et le passage à l'échelle de l'approche proposée à travers des cas d'usage synthétiques. Nous avons d'abord comparé notre approche à une exploration exhaustive par « force brute » pour évaluer son efficacité économique. Les conclusions tirées de cette comparaison ont conduit au développement d'une méthodologie de déploiement améliorée conçue pour une meilleure optimalité. Enfin, l'évaluation du passage à l'échelle de l'approche améliorée a été réalisée. Ces tests ont confirmé que cette solution améliorée s'approche systématiquement de la configuration optimale à coût minimal tout en restant efficace en termes de complexité quand la taille du réseau augmente.

Title: A Methodology for Credit-Based Shaper Deployment in Time Sensitive Networks: Design, Analysis and Optimization

Key words: Time-Sensitive Networking (TSN), Credit-Based Shaper (CBS), Frame-Replication and Elimination for Reliability (FRER), Calcul Réseau (Network Calculus), Interopèrabilité

Abstract: Real-time systems in domains such as aerospace, automotive and industrial automation are increasingly migrating toward Ethernet-based Time-Sensitive Networking (TSN) to meet stringent timing and reliability requirements. While mechanisms like the Credit-Based Shaper (CBS) offer a balance between determinism and complexity, existing literature predominantly assumes a "Full CBS Deployment," where every network switch is TSN-capable. This assumption overlooks the economic and practical reality of heterogeneous networks that must integrate legacy Ethernet devices. Furthermore, the interaction between CBS and reliability mechanisms, specifically Frame Replication and Elimination for Reliability (FRER), remains under-researched, despite evidence that FRER significantly increases traffic burstiness, which is specifically addressed by CBS.
This thesis proposes a novel methodology for the Partial CBS Deployment in time-sensitive networks. We formulate the deployment strategy as an optimization problem aimed at minimizing the number of TSN-enabled devices while ensuring worst-case schedulability requirements using Network Calculus. We introduce a heuristic algorithm that strategically places and configures CBS instances to mitigate traffic burstiness only where necessary, ensuring interoperability with legacy infrastructure. The proposed methodology is validated through a representative industrial use-case, where schedulability constraints were respected while reducing implementation costs.
The methodology is further extended to be FRER-aware, addressing the burstiness increase introduced by such a mechanism. This extended approach neutralizes the adverse effects of redundancy on network latency while minimizing the use of TSN-enabled devices. This extension is validated through the same representative industrial use-case used for the base version of our methodology (FRER-blind). Here, the observations were consistent with our findings for the original approach.
Finally, the thesis evaluates the optimality as well as the scalability of our proposed methodology through synthetic use-cases. We first benchmarked our approach against an exhaustive "brute-force" exploration to assess its optimality. Insights from this comparison led to the development of an improved CBS deployment heuristic designed for better optimality. Lastly, the scalability of such an improved approach was performed. The results confirmed that this enhanced solution consistently approximates the optimal, minimal-cost configuration while remaining computationally efficient under network scale increase.