EPFL

Thèse n° 10 523

Worst-Case Delay Analysis of Time-Sensitive Networks with Network Calculus

Présentée le 15 décembre 2023

Faculté informatique et communications
Laboratoire pour les communications informatiques et leurs applications 2
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Seyed Mohammadhossein TABATABAEE

Acceptée sur proposition du jury

Prof. V. Kuncak, président du jury
Prof. J.-Y. Le Boudec, Prof. P. Thiran, directeurs de thèse
Prof. A. Rizk, rapporteur
Prof. J. Schmitt, rapporteur
Prof. M. Grossglauser, rapporteur

École polytechnique fédérale de Lausanne

2023

To *Mahshid*, my extraordinary mother
To *Mehdi*, my strong father
To *Nasim*, my beloved wife ...

Acknowledgements

First and foremost, I extend my heartfelt thanks to my thesis director, Prof. Jean-Yves Le Boudec. I cannot thank him enough for his invaluable guidance, support, and mentorship throughout my PhD research. His passion for research, dedication to teaching, and commitment to excellence have inspired me and countless others. Working under his supervision and learning from him has been a real honor. As his final PhD student, I can only confirm that he has profoundly impacted the lives of so many students, researchers, and colleagues. I also thank my thesis co-director, Prof. Patrick Thiran, who agreed to be my backup advisor and has made this PhD thesis possible and peaceful.

I thank my PhD defense committee members, Prof. Viktor Kunčak, Prof. Amr Rizk, Prof. Jens Schmitt, and Prof. Matthias Grossglauser, for their time and support through this final step of the PhD process.

Having the best administrative assistants facilitated my life in Lausanne: The LCA2 secretaries, Patricia Hjelt and Angela Devenoge, were extremely helpful in assisting me with settling down and coordinating my work trips. Holly B. Cogliati, a professional editor, was always willing to hollify my papers' English and lend a hand in filling in any missing articles. Regarding technical IT issues, the system administrators, Marc-André Lüthi and Yves Lopes, were my go-to people. I truly appreciate all these individuals' dedication and hard work.

During my time in LCA2, I had the wonderful opportunity to work alongside pleasant colleagues such as Jagdish, Marguerite, Roman, Alaeddine, Ehsan, Ludovic, Stéphan, Plouton, and Louis. We always had interesting discussions during lunch and coffee breaks. And we collaborated to organize course exercises and projects, sharing many enjoyable moments. I thank all the bachelor's and master's students I had the pleasure to work with on fascinating projects: Andrea, Karim, Dubravka, Cyril, Weiran, Chun-Tso, and Edin. Also, I thank my friends outside LCA2 and the Iranian community that mainly formed my outside-the-office life.

I am deeply grateful to my parents, Mahshid and Mehdi, for your unwavering support, particularly during my PhD journey. I could not have accomplished anything without your love, encouragement, and faith in me. I am forever grateful for the opportunities you have given me. Thank you for always believing in me and for your unconditional love. Also, I give many thanks to my super-kind sister, Malihe, and my brothers, Amin and Ehsan, for being the best

siblings I could wish for.

Last but certainly not least, I extend my heartfelt thanks to my loving wife, Nasim. Your unwavering belief in me, patience, and understanding have been the bedrock of my achievements. Your continuous support and selflessness have upheld me during difficult moments, and your companionship has given purpose to this adventure. I appreciate you consistently being by my side when I needed you the most.

Lausanne, July 21, 2023 *Hossein Tabatabaee*

Abstract

Time-sensitive networks, as in the context of IEEE Time-Sensitive Networking (TSN) and IETF Deterministic Networking (DetNet), offer deterministic services with guaranteed, bounded latency in order to support safety-critical applications. In this thesis, we focus on the analysis of time-sensitive networks to address an essential requirement, namely, worst-case delay. Finding the exact worst-case delays is an NP-hard problem that is generally not feasible; therefore, we are interested in bounds on the worst-case delays. To this end, a standard approach is network calculus. It abstracts the service offered by a node by means of a service curve. It then uses service-curve characterizations of the network nodes and arrival curves of flows at their sources and obtains end-to-end delay bounds; an arrival curve is a constraint on the amount of data a flow can send, and it is necessary to obtain finite delay bounds.

First, service-curve characterizations, in some cases, were too simple or non-existent: Round-robin schedulers are widespread, particularly in request balancing in cloud infrastructures, in the Linux Virtual Server scheduling, and in network on chip, and they are known to have efficient implementations. Also, they can be applied to time-sensitive networks, however, they have not been fully analyzed in this context. Interleaved Weighted Round-Robin (IWRR) is a variant of the classic Weighted Round-Robin (WRR) with a smoother service; no prior literature has obtained delay bounds for IWRR. We find the best obtainable strict service curve for IWRR, and we show that delay bounds derived from it are tight for flows of packets of constant size. With IWRR and WRR, the allocated bandwidth to each flow depends on the packet sizes; Deficit Round-Robin (DRR) is a later variant that solves this and provides fair queuing with variable-length packets. We derive the best obtainable strict service curve for DRR, where delay bounds derived from it dramatically dominate all previous works. So far, we have not considered that DRR automatically allocates unused capacity to improve service for other queues. Hence, we obtain delay bounds that remain valid, even if some traffic classes misbehave, but might be overly pessimistic in cases where interfering traffic is limited and behaves as expected. For such cases, we find novel strict service curves for DRR and show that delay bounds derived from them significantly dominate all previous works. Following our work for DRR, others found similar results for WRR and IWRR. End-to-end delay bounds in a DRR network can be obtained using global network analysis with our DRR strict service curve. For the former, Polynomial-size Linear Programming (PLP) is known to provide better bounds and larger stability region compared to its existing alternatives, but it was never applied to DRR

networks. However, this raises dependency loops in networks with cyclic dependencies: On the one hand, our DRR strict service curves rely on traffic characteristics inside the network, which comes as the output of PLP. On the other hand, PLP requires prior knowledge of the DRR service curves. Iterative methods can solve this. However, PLP itself requires making cuts, which imposes other levels of iteration. We propose, PLP-DRR, a generic method for combining all the iterations sequentially or in parallel. We provide the best-known, proven worst-case delay analysis of time-sensitive networks of generic topology with round-robin schedulers, which dramatically dominate all previous works.

Second, for tractability, the arrival curve constraints of flows are often taken to be affine functions. For periodic flows, a common and critical type of traffic in time-sensitive networks, affine arrival curves are known to provide less good bounds than ultimately pseudo-periodic (UPP) arrival curves that precisely capture the periodic behaviors. This is because, in existing tools, handling many periodic flows and UPP curves becomes quickly intractable: When aggregating several UPP curves, the pseudo-period of the aggregate might become extremely large. We propose, FH-TFA, a method that computes finite horizons over which arrival and service curves can be restricted without affecting the end results. This method significantly improves bounds obtained using linear curves while remaining computationally feasible, as we show for industrial networks. FH-TFA has been jointly implemented with RealTime-at-Work. An orthogonal direction for reducing the pessimism of aggregating arrival curve constraints is to use the affine functions but to permit some violation probability. We consider independent periodic flows, and we compute quasi-deterministic and affine arrival curve constraints for their aggregate traffic. The deterministic approach is tight only when all periodic flows are perfectly synchronized, which is highly unlikely in practice and results in an overly pessimistic bound. Our quasi-deterministic arrival curve constraint with a small, non-zero violation probability is considerably smaller than the deterministic one and grows sub-linearly, unlike the deterministic one that grows linearly. Our quasi-deterministic bounds are the first of their kind. We provide the best-known, proven worst-case delay analysis of time-sensitive networks of generic topology with many periodic flows that, while remaining computationally feasible, dramatically reduce the pessimism of existing works.

***Keywords*—** Network calculus, Time-Sensitive Networks, Deterministic Networks, Delay bound, Weighted Round-Robin (WRR), Interleaved Weighted Round-Robin (IWRR), Deficit Round-Robin (DRR), Periodic Flows, Total Flow Analysis (TFA), Polynomial-Size Linear Program (PLP), Quasi-Deterministic, Aggregate Burstiness, Service Curve, Arrival Curve.

Résumé

Les réseaux temps-réels, comme ceux spécifiés par lEEE *Time-Sensitive Networking* (TSN) et lETF *Deterministic Networking* (DetNet), offrent des services déterministes avec une latence garantie et bornée pour prendre en charge des applications critiques. Dans cette thèse, nous nous concentrons sur l'analyse des réseaux temps-réels afin de répondre à une exigence essentielle, à savoir le délai pire-cas. Trouver les délais de pire cas exacts est un problème NP-difficile et généralement non réalisable ; par conséquent, nous nous intéressons aux délais pire-cas bornés. Une approche standard est le calcul réseau. Elle abstrait le service offert par un nœud au moyen d'une courbe de service. Elle utilise ensuite les caractérisations des courbes de service des nœuds du réseau et les courbes d'arrivée des flux à leurs sources pour obtenir des bornes de délai de bout en bout ; une courbe d'arrivée est une contrainte sur la quantité de données qu'un flux peut envoyer, et elle est nécessaire pour obtenir des bornes de délai finies.

Premièrement, les caractérisations des courbes de service sont parfois trop simples ou inexistantes : les ordonnanceurs round-robin sont couramment utilisés, en particulier dans l'équilibrage des requêtes dans les infrastructures de réseau en nuage, dans la planification du serveur virtuel Linux et dans les réseaux sur puce, et sont connus pour avoir des implémentations efficaces. Ils peuvent également être appliqués aux réseaux temps-réels ; cependant, ils n'ont pas été entièrement analysés dans ce contexte. *Interleaved Weighted Round-Robin* (IWRR) est une variante du *Weighted Round-Robin* (WRR) classique avec un service plus régulier. Aucune borne de délai pour l'IWRR n'a été proposée dans la littérature. Nous dérivons la meilleure courbe de service stricte possible pour l'IWRR, et nous montrons que les bornes de délai dérivées de celle-ci sont exactes pour les flux de paquets de taille constante. Avec l'IWRR et le WRR, la bande passante allouée à chaque flux dépend des tailles de paquets ; *Deficit Round Robin* (DRR) est une variante ultérieure qui atténue cela et offre un équitable traitement des files d'attente avec des paquets de longueur variable. Nous dérivons la meilleure courbe de service stricte possible pour le DRR, dont les bornes de délai dérivées dominent largement tous les travaux existants. Jusqu'à présent, nous n'avons pas pris en compte le fait que le DRR alloue automatiquement une capacité inutilisée pour améliorer le service des autres files d'attente ; ainsi, nous avons obtenu des bornes de délai qui restent valables même si certaines classes de trafic se comportent mal, mais qui peuvent être excessivement pessimistes quand le trafic perturbateur est limité et se comporte comme prévu. Pour de tels cas, nous dérivons de nouvelles courbes de service strictes pour le DRR, et nous montrons que les bornes de délai dérivées de celles-ci dominent considérablement tous les travaux existants. À la suite de notre travail sur le DRR, des résultats similaires pour le WRR et l'IWRR ont été obtenu dans la littérature. Dans un réseau DRR, les bornes de délai de bout en bout peuvent être obtenues en utilisant une analyse globale du réseau avec notre courbe de service stricte. *Polynomial-size Linear Program* (PLP) est une analyse globale du réseau qui est connue pour fournir de meilleures bornes et une région de stabilité par rapport à ses alternatives existantes, mais n'a jamais été appliquée aux réseaux DRR.

Cependant, cela crée des boucles de dépendance dans les réseaux avec des dépendances cycliques : d'une part, nos courbes de service strictes pour le DRR reposent sur les caractéristiques du trafic à l'intérieur du réseau, qui sont les résultats de PLP, et d'autre part, PLP nécessite une connaissance préalable des courbes de service du DRR. Les méthodes itératives peuvent être utilisées, mais PLP lui-même nécessite de faire des coupes, ce qui impose d'autres niveaux d'itération. Nous proposons une méthode générique, appelée PLP-DRR, pour combiner toutes les itérations séquentiellement ou en parallèle. Nous améliorons significativement l'état de l'art pour un réseau industriel. Nous avon obtenu la meilleure analyse de délai pire-cas pour un réseau temps-réels quelle que soit la topologie avec des ordonnanceurs round-robin. Nos résultats dominent largement tous les travaux existants.

Deuxièmement, pour des raisons de faisabilité, les contraintes des courbes d'arrivée des flux sont souvent des fonctions affines. Pour les flux périodiques, un type de trafic très utilisé dans le cadre des réseaux temps-réels, les courbes d'arrivée affines sont connues pour fournir des bornes moins bonnes que les courbes d'arrivée ultimement pseudo-périodiques (UPP), qui capturent précisément les comportements périodiques. Cela est dû au fait que, dans les outils existants, la gestion de nombreux flux périodiques et des courbes UPP devient rapidement très complexe : lors de l'agrégation de plusieurs courbes UPP, la pseudo-période de l'agrégat peut devenir extrêmement grande. Nous proposons une méthode, appelée FH-TFA, qui calcule des horizons finis sur lesquels les courbes d'arrivée et de service peuvent être restreintes sans affecter les résultats finaux. Cette méthode améliore considérablement les bornes obtenues à l'aide de courbes linéaires, tout en restant réalisables en termes de calcul, comme nous le montrons pour des réseaux industriels. FH-TFA a été implémenté conjointement avec RealTime-at-Work. Une direction orthogonale pour réduire le pessimisme de l'agrégation des contraintes des courbes d'arrivée consiste à utiliser les fonctions affines mais à autoriser une certaine probabilité de violation. Nous considérons des flux indépendants et périodiques, et nous calculons des contraintes de courbes d'arrivée quasi-déterministes et affines pour leur trafic agrégé. L'approche déterministe est précise uniquement lorsque tous les flux périodiques sont parfaitement synchronisés, ce qui est hautement improbable en pratique et entraîne une borne excessivement pessimiste. Notre contrainte de courbe d'arrivée quasi-déterministe avec très faible probabilité de violation non nulle est considérablement plus petite que la borne déterministe, et elle croît de manière sous-linéaire, contrairement à la borne déterministe qui croît de manière linéaire. Nos bornes quasi-déterministes sont les premières de leur genre dans la littérature. Nous avon obtenu la meilleure analyse de délai pire-cas pour un réseau temps-réels quelle que soit la topologie avec de nombreux flux périodiques. Nos résultats, tout en restant réalisables en termes de calcul, réduisent considérablement le pessimisme des travaux existants.

Mots clés— Calcul réseau, Réseaux temps-réels, Réseaux déterministes, Bornes de délai , Round-robin pondéré (WRR), Round-robin pondéré et entrelacé (IWRR), Round-robin à déficit (DRR), Flux périodiques, Analyse de flux total (TFA), Programme linéaire à complexité polynomial, Quasi-déterministe, Rafales agrégées, Courbe de service, Courbe d'arrivée.

Contents

List of Figures

List of Tables

Acronyms

AFDX Avionics Full-Duplex swithed Ethernet.

ATM Asynchronous Transfer Mode.

AVB Audio Video Bridging.

CBS Credit Based-Shaper.

CPS Cyper-Physical Systems.

CPU Central Processing Unit.

DetNet Deterministic Networking.

DKW Dvoretzky–Kiefer–Wolfowitz.

DRR Deficit Round-Robin.

ELP Exponential-size Linear Programming.

FF-TFA Feed-Forward Total Flow Analysis.

FH-TFA Finite Horizon Total Flow Analysis.

FIFO First-In-First-Out.

FP-TFA FixPoint Total Flow Analysis.

GFP-TFA Generic Fixed Point Total Flow Analysis.

GNN Graph neural Network.

GPC Greedy-Processing Component.

IEEE Institute of Electrical and Electronics Engineering.

IETF Internet Engineering Task Force.

iPLP ineger Polynomial-size Linear Programming.

IWRR Interleaved Weighted Round-Robin.

JSON JavaScript Object Notation.

LANs Local Area Networks.

LP Linear Programming.

LUDB Least Upper Delay Bound.

MILP Mixed-Integer Linear Programming.

NFV Network Function Virtualizations.

NoC Network on Chip.

PBOO Pay Burst Only Once.

PLP Polynomial-size Linear Programming.

PMOO Pay Multiplexing Only Once.

RTaW RealTime-at-Work.

RTC Real-Time Calculus.

SBB Stochastically Bounded Burstiness.

SFA Single Flow Analysis.

SOA State-Of-the-Art.

TAS Time-Aware Shaper.

TFA Total Flow Analysis.

TMA Tandem Matching Analysis.

TSN Time-Sensitive Networking.

UA Ultimately Affine.

UPP Ultimately Pseudo-Periodic.

WRR Weighted Round-Robin.

XML Extensible Markup Language.

Part I
Introduction and Technical Background

1 Introduction

1.1 Context

Time-sensitive networks are a subset of communication networks that offer deterministic services with guaranteed, bounded latency. They support safety-critical applications with delay constraints, where violation can cause catastrophic damages such as death, severe injuries, significant financial loss, and/or harm to the environment [1, Section 10.10]. Examples of time-sensitive networks with such applications are, but not limited to, the following: Automotive networks, where safety-relevant control messages have a delay constraint of 1ms [2]; avionics networks, where parametric data for the fly-by-wire system have a delay constraints of 2ms [3]; and industrial automation, where fault detection in power-line equipment has a delay constraint of 100ms [4]. This contrasts classic communication networks, such as the Internet, that provide statistical guarantees in terms of average delay, bandwidth, and packet loss.

Time-sensitive networks originated in the late twentieth century and were initially used in only a few of industrial sectors, including aerospace and automotive. Today, more applications and industries require deterministic services such as the tactile Internet [5], the industrial Internet of Things [6], electricity distribution [7], and Industry 4.0 [8]. The Institute of Electrical and Electronics Engineering (IEEE) responded to the growing need for deterministic services by creating standards for performance guarantees in multimedia applications. These standards include requirements for flow behavior and the quality of service, as well as mechanisms for scheduling, shaping, and reservation. This effort resulted in the development of the IEEE 802.1BA Audio Video Bridging (AVB) standard [9]. Currently, the IEEE 802.1 Time-Sensitive Networking (TSN) task group [10] is expanding its standardization efforts to include a broader range of applications than what is offered by IEEE 802.1BA AVB. This includes both control-data traffic and multimedia streams over the Link Layer (L2) of the Internet protocol suite [11]. The purpose of the IEEE TSN initiative is to bring together the requirements and mechanisms of existing Ethernet-based solutions for time-sensitive networks, with a focus on interconnecting switches and end systems within Local Area Networks (LANs). The Deterministic Networking (DetNet) working group [12] of the Internet Engineering Task Force (IETF) is working on expanding time-sensitive networking by incorporating the Internet Layer (L3) and utilizing IP packets. This expansion will benefit applications such as electrical utilities, building-automation systems, and industrial machine-to-machine communication [13].

The time-sensitive networks studied in the thesis are FIFO-per-class networks: We assume that flows are grouped into classes, packets inside one class are processed First-In-First-Out (FIFO), and classes are

isolated using schedulers. Flows with different worst-case delay requirements coexist in time-sensitive networks. To support such requirements, flows are grouped into some classes, and the service received by the flows of each class is isolated by means of a scheduling policy implemented at network nodes; the scheduling policy describes the algorithm or set of algorithms in a system that decides which packet to serve among those waiting for service. There exists a system-level and a class-level scheduling policy: The former decides which class to serve among those waiting, and the latter decides which packet to serve among those of the class waiting for service; throughout this thesis, this is the FIFO policy.

1.1.1 Worst-Case Delay Guarantees for Time-Sensitive Networks

Time-sensitive networks require deterministic guarantees on worst-case delay, worst-case delay-jitter (defined as the difference between worst-case and best-case delays), zero congestion-loss, jitter, in-order packet delivery, and seamless redundancy [14, 15, 16, 17, 18, 19]. The main focus of this thesis is on the worst-case delays: For a flow, the maximum delay of all non-lost packets sent by the flow during its lifetime; this is one of the most common requirements across time-sensitive applications. The problem of finding the exact worst-case delays in a network setting is known to be NP-hard and is not feasible in general [20, 21]: For tree networks with First-In-First-Out (FIFO) multiplexing nodes, finding the exact worst-cases is super-exponential and applicable on only very small sized networks [20, 22]; for non-tree networks or networks with cyclic dependencies, there is no approach that finds the exact worst cases, even when node scheduling is as simple as can be (FIFO). Hence, the exact worst cases are unknown. An alternative is to find achievable delays by means of simulation or real-life measurements [23, 24]. However, the main limitation of this approach is the difficulty in detecting rare events with low probabilities, and thus the measured delays serve only as lower bounds of the unknown worst-case delay and do not provide deterministic guarantees. Therefore, to validate a deterministic service, we require an upper bound on the worst-case delay.

Obtaining upper bounds on the worst-case delay requires the use of deterministic approaches. Several deterministic approaches have been used for time-sensitive networks: The model-cheeking approach [25] verifies system properties by analyzing states and transitions. Several time-sensitive networks had their timing properties verified by using it [26, 27]. Model-checking approaches can determine the exact worst-case delays, but is only applicable on very small-sized networks as the number of possible network states grows exponentially. The trajectory approach [28] involves finding trajectory scenarios where a flow experience the worst interference from all other interfering flows, and it was used for Avionics Full-Duplex swithed Ethernet (AFDX) [29, 30, 31]. However, finding such scenarios is a hard problem and impractical, because there are numerous interference patterns and diverse network systems. Holistic approaches [32, 33, 34, 35, 36, 37] consider the inter-dependencies and interactions among various system components, tasks, and resources. They struggle to find end-to-end delay bounds in networks with asynchronous system-level scheduling (e.g., round-robin schedulers, credit-based schedulers). Network calculus [38, 39, 40] is a mathematical framework with a set of tools and results for computing delay and backlog bounds. It has been useful for modeling time-sensitive networks with complex topologies and technologies, and it has been applied in various industries and applications: For AFDX in [41, 42, 43], for industrial Internet of Things in [44, 45, 46], for Cloud Computing in [47, 48], for Cyper-Physical Systems (CPS) in [49, 50, 51], for Multimedia Streaming in [52, 53], etc. We select network calculus as the deterministic approach used in this thesis.

1.1.2 Network Calculus: Arrival Curves, Service Curves, and FIFO-Per-Class Heuristics

Network calculus abstracts the service offered by a network node to flows of a given class by means of a (per-class) service curve. Per-class service curves have been vastly analyzed for some scheduling policies: Static Priority [17, Section 8.6.8.1] in [38, Section 6.2.1][54, Sections 7.3.2, 8.2.1]; Credit Based-Shaper (CBS) [17, Section 8.6.8.2] in [55, 56, 57, 58]; Time-Aware Shaper (TAS) [17, Sections 8.6.8.3-4, 8.6.9] in [54, Section 8.2.5] [59]. Strict service curves are a special kind of service curve that are frequently used. A strict service curve is a function that lower-bounds the service offered by the system in its backlogged periods.

Consider a network element and a class of interest. A bound on the worst-case delay is obtained by combining the per-class service curve with an arrival curve for the class of interest. An arrival curve is a constraint on the amount of data observed for traffic; such a constraint is necessary for the existence of a finite delay bound. In a network setting, network calculus limits the amount of data a flow can send at the source by some arrival curves. Finding accurate arrival curves for flows at their sources is generally less challenging compared to service curves; still, a recent work improved arrival curve constraints for when packet-level information is available [60].

Then, for flows of a given class, network calculus analyzes the FIFO-per-class network: It uses the per-class service curve characterization at network nodes and arrival curve constraints of flows at their sources and computes end-to-end delay bounds. Several heuristics have been proposed for the analysis of FIFO-per-class networks: Total Flow Analysis (TFA) [61, 62, 63, 64], Single Flow Analysis (SFA) [62, Section 3.3], Pay Multiplexing Only Once (PMOO) [62, Section 3.4][65], Least Upper Delay Bound (LUDB) [22, 66], Flow Prolongations [67], Exponential-size Linear Programming (ELP) [20], Polynomial-size Linear Programming (PLP) [68], etc. Among these, in this thesis, we consider TFA and PLP: They are the only known methods that can be applied to FIFO-per-class networks with cyclic dependencies. For time-sensitive networks, cyclic dependencies are linked to certain primary properties, such as improving availability and decreasing reconfiguration effort, hence they are important and cannot be ignored [69, 70].

It is known that TFA is outperformed by PLP that always provides delay bounds better than or equal to those of TFA and, at high network utilization, often converges when TFA does not. TFA is still of interest, as it is much simpler and more tractable than PLP. Also, by design, PLP uses delay bounds obtained by TFA (if available) as a constraint in all its linear programs. Note that other methods, such as LUDB [22] and flow prolongations [67], also tend to dominate TFA, however, unlike PLP, they do not apply to generic topologies.

1.2 Gaps in Worst-Case Delay Analysis of Time-Sensitive Networks with Round-Robin Schedulers

One of the first use of round-robin scheduling in the network context appeared in [71], with a fairness objective, i.e., a fair way to share the bandwidth among sessions. It is also mentioned in [72] as a way to implement “fair queueing”. Round-robin schedulers have been applied in Ethernet [73, Sec. 8.6, Sec. 8.6.8.3, Sec. 37], in request balancing in cloud infrastructures [74], in the Linux Virtual Server scheduling [75], in network of chip [76], etc. Also, round-robin schedulers are a great candidate for network slicing that is a natural solution to simultaneously accommodate, over a common network

infrastructure, the wide range of services, e.g., in 5G networks with Network Function Virtualizations (NFV) [77, 78, 79]. They have been widely used as they have low complexity and very efficient implementations exist [80, 81, 82, 83]. Round-robin schedulers can be applied to time-sensitive networks, however, they have been overlooked and not been fully analyzed in the context of time-sensitive networks.

1.2.1 Non-Existent Service Curve Characterization for Interleaved Weighted Round-Robin (IWRR)

An important variant of round-robin schedulers is Weighted Round-Robin (WRR). With WRR, the capacity is shared among several queues by giving each of them a weight, which is a positive integer, and by providing more service to those with larger weights. Specifically, queues are visited one after the other, and when a queue has an emission opportunity, a number of packets equal to the weight allocated to the queue can be served consecutively, which leads to a bursty service. Interleaved Weighted Round-Robin (IWRR) mitigates this effect [84]. IWRR spreads out emission opportunities of each queue, which is expected to result in a smoother service and lower worst-case delays.

However, no prior literature exists for worst-case delay bounds with IWRR. The network calculus approach was applied to WRR in [40, Sec. 8.2.4], where a strict service curve is obtained. However, compared to WRR, the interleaving in IWRR makes the analysis more difficult, and the method of proof in [40, Sec. 8.2.4] cannot easily be extended.

1.2.2 Too Simple Service Curve Characterizations for Deficit Round-Robin (DRR)

Although WRR and IWRR were originally designed in the context of Asynchronous Transfer Mode (ATM) [85] with constant-size packets to share the bandwidth in proportion to allocated weights, they have been applied to networks with variable-length packets. In such cases, the allocated bandwidth to each queue depends on not only the weights but also on the packet sizes. This is not desirable, as the intention of the weights is to control the allocated bandwidth to each queue; however, they do not entirely control this as the packet sizes interfere. Deficit Round-Robin (DRR) is a later variant that solves this and achieves fair queuing in the presence of variable-length packets. With DRR, every queue is associated with a static number, called quantum. Queues are visited one after the other, and at every visit, they receive service (measured in bits for communication networks, in seconds for task processing systems) up to the quantum value. Tasks or packets are of variable sizes, and it may happen that, during one visit of the server, there remains at least one task or packet in the queue that cannot be served because the unused part of the quantum is positive but not large enough. In such a case, the unused part of the quantum (called the residual deficit) is carried over to the next round. DRR shares resources flexibly (the amount of service reserved for one queue is proportional to its quantum) and efficiently (when a queue is idle, the server capacity is available to other queues). It is widely used as it has low complexity and very efficient implementations exist [80].

Worst-case delay bounds for DRR were obtained in [86, 87, 88] by using various ad-hoc analyses. These results were improved in [89], where the authors obtain a strict service curve for DRR; this strict service curve is too simple and does not account for the details of DRR hence calls for improvement.

Round-robin schedulers are efficient, which means that, when a queue is idle, the server capacity is available to other queues. The service curves we presented in the last paragraph do not take this

into account as they do not make any assumptions on the interfering traffic; thus, on the one hand, delay bounds derived from them are valid even if the interfering traffic misbehaves, and on the other hand, they might be pessimistic, when all time-sensitive traffic behaves as expected. In time-sensitive networks, some or all interfering traffic is deterministic and, in normal operation, is limited at the source by an arrival curve constraint. There is also interest in obtaining proven bounds under normal conditions when all time-sensitive traffic satisfies its source constraints. For such cases, two improvements were proposed. The former uses a correction term derived from a semi-rigorous heuristic [90]. The latter rigorously derives convex strict service curves for DRR that account for the arrival curve constraints of the interfering traffic [91]. However, these strict service curves are too simple and do not account for the details of DRR hence call for improvement.

1.2.3 Loose End-to-End Delay Bounds in Time-Sensitive Networks with DRR

As explained in Section 1.1.2, finding delay bounds in a DRR network requires incorporating DRR strict service curves with a FIFO-per-class network analysis. One aspect of improving delay bounds is by improving service curve characterizations. Another aspect is to employ better heuristics for the analysis of FIFO-per-class networks. PLP is known to provide better bounds and stability region compared to its existing alternatives but has never been applied to DRR networks. A straightforward application of PLP is to use DRR strict service curves that have no assumption on the interfering traffic. They depend only on the assigned quantum and maximum packet size of every class hence can be computed for all classes at all nodes a priori to PLP. Thus, per-class networks are independent and can be analyzed separately (i.e., the network is sliced into some per-class networks), and one instance of PLP can be run per-class to obtain bounds. However, the combination of PLP with DRR strict service curves that account for the interfering traffic is far from straightforward: Using PLP to analyze DRR networks requires introducing the DRR strict service curve into the PLP procedure. PLP uses internal variables that the computation of which depends on the DRR strict service curves; the DRR strict service curves rely on traffic characteristics inside the network, which comes as the output of PLP. Iterative methods can solve this, however, PLP itself requires making cuts, which imposes other levels of iteration.

1.3 Gaps in Worst-Case Delay Analysis of Time-Sensitive Networks with Many Periodic Flows

The development of industrial automation requires timely and accurate monitoring of the status of the network. In time-sensitive networks, a common assumption for critical types of traffic is that devices send packets periodically. This makes periodic flows a common and critical type of traffic in time-sensitive networks [2, 3, 4]

1.3.1 Too Simple Arrival Curve Constraints for Periodic Flows

Periodic flows are known to be constrained by an Ultimately Pseudo-Periodic (UPP) arrival curve that precisely captures the periodic behaviors. Tools such as RTaW [92], Nancy [93], DiscoDNC [94], etc. use infinite precision arithmetic (with rational numbers) and implement UPP curves; UPP curves have a transient part at the beginning, followed by a pseudo-periodic pattern. UPP curves are of interest in practice as they have a finite representation, and they capture periodic behaviors. However, when aggregating several UPP curves, the pseudo-period of the aggregate function might become extremely

large. Furthermore, the required memory to store the aggregate function might explode, as the number of segments required to describe the aggregate function quickly grows. An early example of where this issue occurs is the avionic onboard communication system analyzed in [95]. In fact, for tractability, arrival curve constraints of periodic flows are often taken to be affine functions. This results in less good delay bounds hence calls for improvements. Also, in networks with cyclic dependencies, there is no method in the existing works that provides proven delay bounds with arrival curves and service curves of generic shapes.

1.3.2 Pessimism in Arrival Curve Aggregation for Periodic Flows

As already mentioned, in time-sensitive networks, a common assumption for critical types of traffic is that devices send packets periodically. These packets are aggregated and forwarded to the controller. Characterizing this aggregate traffic is then crucial for effective resource management. Indeed, an arrival curve constraint can be obtained for the aggregate traffic by aggregating the arrival curve of individual flows. However, such arrival curve superposition is known to be pessimistic: It grows linearly with the number of flows [96]. Also, this is tight only when all periodic flows are perfectly synchronized, which is highly unlikely in practice and results in an overly pessimistic bound. An orthogonal solution to our discussion in the previous section is still to use affine arrival curves but permit some violation probability. Probabilistic versions of network calculus (known as Stochastic Network Calculus) have emerged, and their aim is to compute performances when a small violation probability is permitted. By using probabilistic tools, such as moment-generating functions [97] or martingales [98], recent works mainly compute probabilistic arrival curve constraints for the aggregate traffic with small violation probability at an arbitrary point in time [97, 99, 100, 101, 102, 103]. In time-sensitive networks, we are interested in the probability that a bound is not violated during some interval (e.g., the network's lifetime), not just one arbitrary point in time. Existing works do not provide such information; the violation probability of a bound being small at one arbitrary point in time does not imply that the probability that the bound is never violated during a period of interest is small. In fact, there would likely be some violations.

1.4 Contributions and Roadmap

The details of our contributions in this thesis are as follows:

1. We provide the first-ever delay bounds for IWRR. We find a strict service curve for IWRR, and show that it is the best possible one. We find that delay bounds derived by it are tight (i.e., worst-case) for flows of constant packet sizes; we show similar results for the strict service curve of classic WRR that was previously published. We show that our IWRR strict service curve dominates the WRR strict service curve, hence it dominates the delay bounds. Details are presented in Chapter 3. The content of this chapter was published in [104, 105].
2. We find a novel strict service curve for DRR that we show is the best obtainable one. Also, we introduce a novel method that obtains better strict service curves for DRR when the interfering traffic behaves as expected and is constrained by some arrival curve constraints. Delay bounds derived by our strict service curves are significantly smaller than all previous works. We also show that the first attempt in previous works to obtain delay bounds in such cases is incorrect. Following our research, others provide similar delay bounds for WRR and IWRR [106]. Details are presented in Chapter 4. The content of this chapter was published in [107, 108].

3. We provide a method, called PLP-DRR, for the worst-case delay analysis of DRR network: It combines our DRR strict service curves (that account for the interfering traffic) and PLP in a novel way. We find very significant improvements, for an industrial network, in terms of delay bounds and stability region compared to the previous works. The combination is far from trivial, as there are dependencies in different levels: PLP uses DRR strict service curves and improves the traffic characterization inside the network; and this results in better DRR strict service curves, which in return improves the output of PLP. PLP also uses some internal variables that have two-way dependencies between both DRR strict service curves and the traffic characterization inside the network. Hence, there are many dimensions that play together, and a simple iterative method cannot solve this issue. We invent a generic framework that uses a distributed computing model with shared memory, where individual improvements can be applied in any order. We theoretically show that any execution provides the same valid bounds, regardless of the order in which the individual improvements are applied. We design two concrete implementations of the method, with parallel for-loops, that are efficient in terms of run times. Also, when applying PLP to DRR, we make two further improvements to the existing PLP. The former computes improved arrival curve constraints for the aggregate of flows. The latter enables us to use non-convex service curves in PLP and obtains better delay bounds. Details are presented in Chapter 5. The content of this chapter was submitted [109].

4. We develop and validate FH-TFA, an algorithm that provides delay bounds for time-sensitive networks with generic topology, and generic arrival and service curves that are implemented as UPP curves. We give a numerical application to real, industrial cases, provided by industrial partners of RealTime-at-Work (RTaW), and we observe that bounds obtained with FH-TFA and UPP curves are considerably less than previous works. To obtain FH-TFA, we first generalize the theory of existing versions of TFA to arrival curves and service curves of generic shapes for networks with generic topology. We then replace the original, UPP arrival and service curves with some simpler curves that are accurate enough not to affect the end results but also simple enough to remain tractable in the existing tools. FH-TFA has been jointly implemented with RTaW. Details are presented in Chapter 6. The content of this chapter was published in [110].

5. We find quasi-deterministic arrival curve constraints for the aggregate traffic of independent, periodic flows. We use affine arrival curve constraints, but we permit some violation probability and obtain an arrival curve constraint for the aggregate that is valid at all times with large probability. Our quasi-deterministic bounds are the first of their kind. For the homogeneous case, we obtain a closed-form expression that, for a non-zero violation probability, provides a considerably smaller arrival curve for the aggregate than the deterministic one; it grows sub-linearly, unlike the deterministic one that grows linearly. For the heterogeneous case, we obtain a bound by grouping flows into homogeneous sets and combining the bounds obtained for each set using a convolution bounding technique. Details are presented in Chapter 7. The content of this chapter was submitted [111].

6. *Side contribution:* We present Saihu, a Python interface that integrates the three most frequently used worst-case network analysis tools: xTFA, which implements the method of TFA; DiscoDNC, which implements the methods of TFA, LUDB, PMOO, SFA, etc.; and Panco, which implements the methods of TFA, SFA, PLP and ELP. Saihu enables users to define a network once and execute all tools in a single shot, without any modification. It is open-source and publicly available [112]. Details are presented in Appendix A. The content of this chapter was submitted [113].

2 Technical Background

In this chapter, we provide the necessary and common background of the rest of the thesis. We provide a summary of network calculus in Section 2.1. We also present pseudo inverse functions in Section 2.2. A summary of notation and symbols used throughput this thesis are given in Section 2.3.

2.1 Network Calculus

Network calculus [38, 39, 40] is a theory for obtaining bounds on the worst-case delay and backlog of communication networks. Le Boudec and Chang concurrently [114, 115] developed its framework based on the seminal work of Cruz [116, 117]. In this section, we first provide a summary of the basic concepts in network calculus; then, we present the main theorems for obtaining backlog and delay bounds. We also provide the network calculus model for time-sensitive networks. We give a list of worst-case delay analysis tools. Lastly, we give a list of methods and techniques to compute end-to-end delay bounds within a network. Readers who might prefer a video tutorial or those who are new to concepts of network calculus are strongly invited to watch the following tutorial [118].

In this thesis, we often use wide-sense increasing functions. Let $\mathscr{F}$ denote the set of wide-sense increasing functions $f : \mathbb{R}^+ \mapsto \mathbb{R}^+ \cup \{+\infty\}$, where $\mathbb{R}$ is the set of real numbers; we say f is wide-sense increasing if and only if $\forall 0 \leq s \leq t,\ f(s) \leq f(t)$.

2.1.1 Main Concepts of Network Calculus

2.1.1.1 Cumulative Functions

Consider a system S and a flow f through S. The system can be a single queue or a network of queues. Network calculus, for convenience, describes the data flow by means of the cumulative function arrival (resp. departure) function A_f (resp. D_f), where $A_f(t)$ (resp. $D_f(t)$) is the number of bits arrived (resp. departure) for flow f between times 0 and t. By convention, we take $A_f(0) = D_f(0) = 0$, which means time 0 represents a time origin where no data has been sent before $t = 0$. By definition, cumulative functions are wide-sense increasing. In the rest of this thesis, we assume that the time and the amount of data are continuous quantities unless otherwise specified (See Fig 2.1).

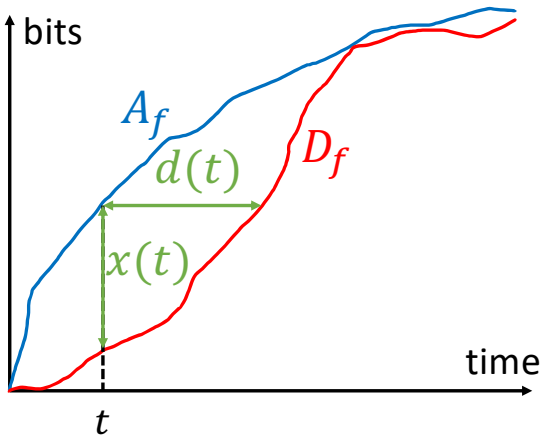


Figure 2.1: Representing the arrival and departure by cumulative functions results in conveniently finding the delay and backlog: $x(t)$ is the backlog at time t (if the system is causal and lossless) and is the vertical distance between the cumulative arrival function A_f and cumulative departure function D_f at time t. Also, the horizontal distance $d(t)$ between the functions is the delay that the bit arriving at t experiences in the system (if the system is causal, lossless, and FIFO).

2.1.1.2 Backlog and Delay

The choice of representing the arrival and departure of a flow by cumulative functions results in conveniently finding the delay and backlog: If system S is causal (i.e, the system does not produce or duplicate any data internally), $x(t) = A_f(t) - D_f(t)$ is the amount of data that has entered, but not yet left, the system up to time t; if system S is lossless (i.e., the system does not lose any data) as well, it follows that $x(t)$ is the amount of data inside the system at t, i.e., backlog at time t. Indeed, the backlog at time t is the vertical distance between the cumulative arrival function A_f and cumulative departure function D_f at time t (see Fig. 2.1). Also, if system S is FIFO, it follows that the horizontal distance $d(t)$ between the functions is the delay that the bit arriving at t experiences in the system (see Fig. 2.1). It follows that given the cumulative arrival function A_f and cumulative departure function D_f, the worst-case delay and backlog can be obtained by finding the horizontal and vertical deviation between A_f and D_f; the definitions of horizontal and vertical deviation are as follows:

Definition 2.1 (Horizontal deviation hDev)**.** *For $f, g \in \mathcal{F}$, the horizontal deviation between f and g is* $hDev(f,g) \stackrel{def}{=} \sup_{s\geq 0}\{\inf\{d \geq 0 | f(s) \leq g(s+d)\}\}$.

Definition 2.2 (Vertical deviation v)**.** *For $f, g \in \mathcal{F}$, the vertical deviation between f and g is* $vDev(f,g) \stackrel{def}{=} \sup_{s\geq 0}\{f(s) - g(s)\}$.

The horizontal (resp. vertical) deviation between f and g can be visually obtained by finding the longest horizontal (vertical) line between the two functions.

Note that the pair of (A_f, D_f) is only one instance of many possible arrival and departure functions for flow f through system S; the sources generate the packets at random time instants, thus A_f and D_f are not known and cannot be used to compute the worst-case backlog or delay. We are interested in delay and backlog bounds that are valid for all acceptable pairs of (A_f, D_f). To do so, network calculus relies on arrival and service curves.

2.1.1.3 Arrival Curves

As mentioned, the cumulative arrival function A_f is not known; network calculus counterpart this by limiting the amount of data that a flow can send by means of an arrival curve.

Definition 2.3 (Arrival Curve)**.** *We say that a flow f, with the the cumulative arrival function A_f, is constrained by an arrival curve $\alpha_f \in \mathscr{F}$ if and only if*

$$\forall 0 \leq s \leq t,\ A_f(t) - A_f(s) \leq \alpha_f(t-s) \tag{2.1}$$

An arrival curve α_f limits the amount of data observed for a flow at any *time interval*, and it takes as input the duration of a time interval rather than the absolute times. An arrival curve α_f can always be assumed to be sub-additive, i.e., to satisfy $\forall s,t,\ \alpha_f(t) + \alpha_f(s) \geq \alpha_f(t+s)$. Otherwise, it can be replaced by its sub-additive closure (see [54, Section 2.2]). Two frequently used arrival curves are $\gamma_{r,b}$, a token-bucket function with rate r and burst b, and $\nu_{p,b}$, a stair function defined below. They are sub-additive.

Definition 2.4 (Token-bucket function $\gamma_{r,b}$)**.** *$\forall r, b \geq 0$, the token-bucket function $\gamma_{r,b} \in \mathscr{F}$ is defined by (see Fig. 2.2)*

$$\gamma_{r,b}(t) \overset{def}{=} \begin{cases} 0 & \text{if } t = 0 \\ rt + b & \text{if } t > 0 \end{cases} \tag{2.2}$$

Definition 2.5 (Stair function $\nu_{p,b}$)**.** *$\forall b \geq 0$ and $p > 0$, the stair function $\nu_{p,b} \in \mathscr{F}$ is defined by (see Fig. 2.2)*

$$\nu_{p,b}(t) \overset{def}{=} b\lceil \frac{t}{p} \rceil \tag{2.3}$$

Arrival curves are not unique; indeed, a flow that is constrained by an arrival curve α_f is also constrained by any α'_f such that $\alpha'_f \geq \alpha_f$. Sometimes for tractability, arrival curves are replaced by simpler upper bounds. For example, a periodic flow with period p that sends packets of size b is constrained by a stair arrival curve $\nu_{p,b}$; however, for tractability, it can be replaced by a token-bucket arrival curve $\gamma_{r,b}$ with $r = \frac{b}{p}$ (see Fig. 2.2 and Chapter 6 for more details).

It might also happen that we know several non-dominated arrival curves for a flow; if α_f and α'_f are two arrival curves for flow f, then a better arrival curve can be obtained by $\alpha_f \otimes \alpha'_f$. For example, if a flow, with arrival curve α and a maximum packet size $l^{\max}$, arrives on a link with a rate c, a better arrival curve for this flow at the output of the link is the min-plus convolution of α and the function $t \mapsto l^{\max} + ct$; this is known as line-shaping (also known as grouping)(see Fig. 2.6).

2.1.1.4 Service Curves

As mentioned, the cumulative arrival and departure functions A_f and D_f are not known; by means of an arrival curve, the former is constrained. Then, a service curve abstracts the service offered by the system and models the system.

Definition 2.6 (Service Curve)**.** *Consider a system S and a flow f through S with cumulative arrival and departure functions A_f and D_f and let $\beta \in \mathscr{F}$. We say that the system S offers β as a service curve to the flow if*

$$D_f \geq A_f \otimes \beta \tag{2.4}$$

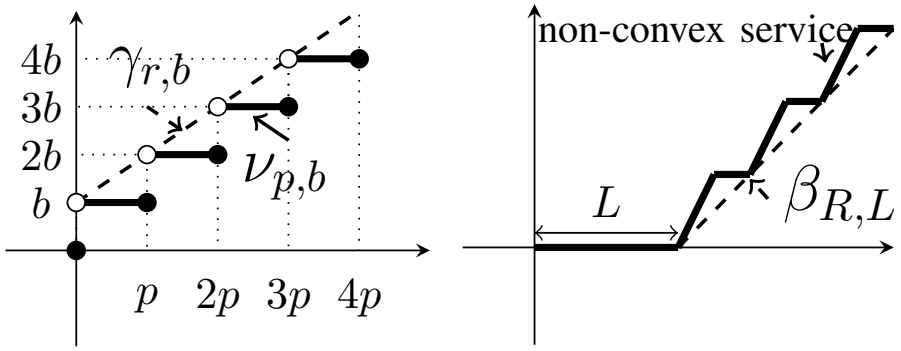
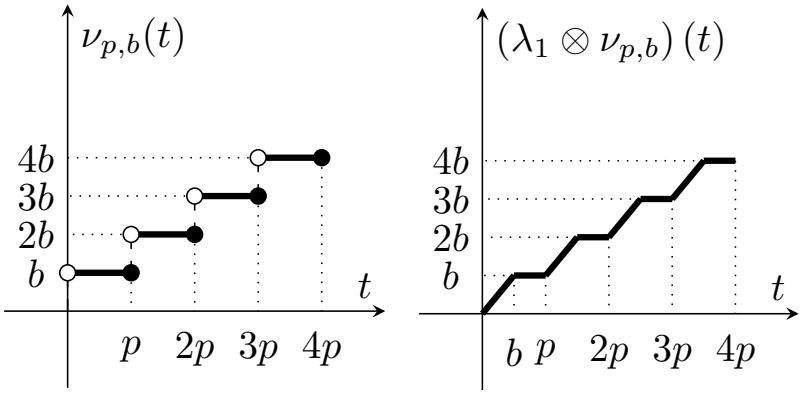


Figure 2.2: Left: the stair function $\nu_{b,p} \in \mathscr{F}$ defined for $t \geq 0$ by $\nu_{b,p}(t) = b\left\lceil \frac{t}{p} \right\rceil$ and token-bucket function $\gamma_{r,b} \in \mathscr{F}$ defined for $t > 0$ by $\gamma_{r,b}(t) = b + rt$ and for $t = 0$ by $\gamma_{r,b}(0) = 0$ (in the figure, we have $r = \frac{b}{p}$). Right: a non-convex service curve and a rate-latency $\beta_{R,L} \in \mathscr{F}$ that lower bounds it with $\beta_{R,L}(t) = \max(0, R(t-L))$.

where, the symbol $\otimes$ denotes the min-plus convolution, defined for arbitrary functions $f, g \in \mathscr{F}$ by

$$(f \otimes g)(t) \stackrel{def}{=} \inf_{0 \leq s \leq t} \{f(t-s) + g(s)\}. \tag{2.5}$$

This often means that for every $t \geq 0$ there exists some $s \leq t$ such that $D_f(t) \geq A_f(s) + \beta(t-s)$. Similar to arrival curves, service curves are defined for time intervals rather than absolute times. See Fig. 2.3 for an example of min-plus convolution used in this thesis.



(a) $\nu_{p,b}$ (b) $(\lambda_1 \otimes \nu_{p,b})$

Figure 2.3: Left: the stair function $\nu_{p,b} \in \mathscr{F}$ defined for $t \geq 0$ by $\nu_{p,b}(t) = b\left\lceil \frac{t}{p} \right\rceil$. Right: min-plus convolution of $\nu_{p,b}$ with the function $\lambda_1 \in \mathscr{F}$ defined by $\lambda_1(t) = t$ for $t \geq 0$. When $b \leq p$, the discontinuities are smoothed and replaced with a unit slope.

Service curves are not unique; indeed, a system that offers a service curve β, also offers any service curves β' such that $\beta' \leq \beta$. Sometimes for tractability, service curves are replaced by simpler lower bounds. For example, a non-convex service curve can be replaced by a rate-latency function that lower-bounds it (see Fig. 2.2 and see Chapters 3 and 4 for more details.).

Definition 2.7 (Rate-latency function $\beta_{R,T}$)**.** *$\forall R, T \geq 0$, the rate-latency function $\beta_{R,T} \in \mathscr{F}$ is defined by (see Fig. 2.2)*

$$\beta_{R,T}(t) \stackrel{def}{=} R[t-T]^+ \tag{2.6}$$

where we use the notation $[x]^+ = \max(0, x)$.

A service curve β can always be assumed to be non-negative and non-decreasing, otherwise, it can be replaced by its non-negative and non-decreasing closure.

Definition 2.8 (Non-decreasing and non-negative closure $[f]^+_\uparrow$)**.** *The non-decreasing and non-negative closure $[f]^+_\uparrow$ of a function $f : \mathbb{R}^+ \to \mathbb{R} \cup \{+\infty\}$ is the smallest function in $\mathscr{F}$ that upper bounds f (see Fig. 2.4) and is given by*

$$[f]^+_\uparrow(t) \stackrel{def}{=} \sup_{s \leq t} [f(s)]^+ \tag{2.7}$$

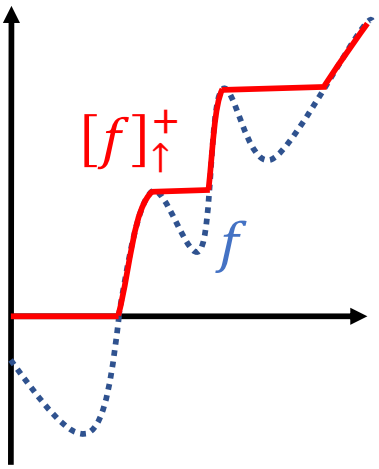


Figure 2.4: A function f and its non-decreasing and non-negative closure $[f]^+_\uparrow$, which is the smallest non-negative and non-decreasing function that upper bounds f.

A special kind of service curve is a *strict* service curve that lower-bounds the service offered by the system in its backlog periods:

Definition 2.9 (Backlogged periods)**.** *Consider a system S and a flow f through S with cumulative arrival and departure functions A_f and D_f. An interval $(s,t]$ is a backlogged period if $A_f(\tau) > D_f(\tau)$ for all τ such that $s < \tau \leq t$.*

For a backlogged period $(s,t]$, $u = \sup_{v \leq t}\{D_f(v) = A_f(v)\}$ is the start of this the beginning of this backlogged period, and indeed $u \leq s$ and $(u,t]$ is also a backlogged period.

Definition 2.10 (Strict Service Curve)**.** *We say that system S offers a* strict *service curve $\beta \in \mathscr{F}$ to the flow if, whenever $(s,t]$ is a backlogged period*

$$D_f(t) - D_s(s) \geq \beta(t-s) \tag{2.8}$$

If β is a strict service curve, then it is a service curve, but the converse is not always true [38, Sec. 1.3]. A frequently used service curve is the rate-latency function $\beta_{R,T}$. Saying that a system offers a

service curve $\beta_{R,T}$ to a flow expresses that the flow is guaranteed a service rate R, except for possible interruptions that might impact the delay by at most T. Saying that a system offers a *strict* service curve $\beta_{R,T}$ to a flow expresses that the flow is guaranteed a service rate R, except for possible interruptions that might not exceed T in total per backlogged period.

A strict service curve β can always be assumed to be super-additive, i.e., to satisfy $\forall s,t,\ \beta(t)+\beta(s) \leq \beta(t+s)$. Otherwise, it can be replaced by its super-additive closure [40, Prop. 5.6]. Note that this is not true for non-strict service curves. Also, if β and β' are two strict service curves offered by S, then S also offers $\max(\beta,\beta')$ as a strict service curve.

2.1.2 Network Calculus Bounds

Theorem 2.1 (Network Calculus Three Bounds)**.** *Assume that a flow f, constrained by arrival curve α_f, traverses a system S that offers a service curve β to the flow. Assume that system S is causal, lossless, and FIFO.*

1. *The delay of flow f is upper bounded by $hDev(\alpha_f,\beta)$;*
2. *The backlog of flow f inside S is upper-bounded by $vDev(\alpha_f,\beta)$;*
3. *Flow f is contained by an arrival curve $\alpha_f^* = \alpha_f \oslash \beta$ at the output of the system.*

In the above, the horizontal deviation hDev and vertical deviation vDev are defined in definitions 2.1, 2.2. The symbol $\oslash$ denotes the min-plus deconvolution, defined for arbitrary functions $f,g \in \mathscr{F}$ by

$$(f \oslash g)(t) \stackrel{def}{=} \sup_{0\leq s}\{f(t+s)-g(s)\} \tag{2.9}$$

See Fig. 2.6 for illustration. The computation of $\mathrm{hDev}(\alpha_f,\beta)$ and α_f^* can be restricted to a sufficient horizon t^*: Only values of $\alpha_f(t)$ and $\beta(t)$ for $t \in [0,\ t^*]$ with $t^* \geq \inf_{s>0}\{\alpha_f(s) \leq \beta(s)\}$ are required (see Fig. 2.6) [40, Prop. 5.13]. In Chapter 6, we will provide more details on the sufficient horizon.

Let us apply Theorem 2.1 on an example: Consider a flow f that crosses n systems S_1 and S_2 in sequence. Assume that S_i offers to flow f the service curve β_i with $i = 1:2$. Also, assume that flow f is constrained by an arrival curve α_f at the input of system S_1 (see Fig. 2.5). Then, at S_1 we apply Theorem 2.1 and obtain $h(\alpha_f,\beta_1)$ a valid delay bound for the first server (item 1); also, we find $\alpha_f \oslash \beta_1$, an arrival curve for the flow at the output of the server (item 3) which can be used as the input to next server. Finally, an end-to-end delay bound can be obtained for the flows by summing the delay bound experienced at each system:

$$d_f^{\mathrm{e2e}} = \mathrm{hDev}(\alpha_f,\beta_1) + \mathrm{hDev}(\alpha_f \oslash \beta_1,\beta_2) \tag{2.10}$$

We can improve on this delay bound by concatenating the service offered by the systems.

Theorem 2.2 (Concatenation of systems)**.** *Consider a flow f that crosses systems S_1 and S_2 in sequence. Assume that S_i offers to flow f the service curve β_i with $i = 1:2$. Then, the concatenation of S_1 and S_2 offers to flow f the service curve $\beta_1 \otimes \beta_2$, where $\otimes$ is the min-plus convolution defined in* (2.5)*.*

By Theorem 2.2 we obtain

$$d_f^{\mathrm{e2e}\prime} = \mathrm{hDev}(\alpha_f,\beta_1 \otimes \beta_2) \tag{2.11}$$

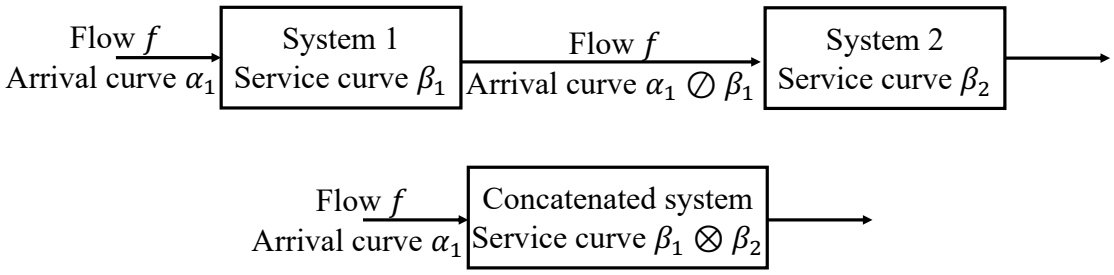


Figure 2.5: A flow f that crosses two systems S_1 and S_2 in sequence. By Theorem 2.2, the concatenation of S_1 and S_2 offers to flow f the service curve $\beta_1 \otimes \beta_2$; then, by one application of Theorem 2.1, an end-to-end delay bound for flow f is obtained. This always dominates the end-to-end delay bound for flow f obtained by the successive application of Theorem 2.1. This effect is known as Pay Burst Only Once (PBOO).

$d_f^{\mathrm{e2e}\prime}$ is always better than d_f^{e2e}, obtained by the successive application of Theorem 2.1. This effect is known as *Pay Burst Only Once (PBOO)*.

2.1.2.1 Packetizer

Theorem 2.1 requires abstracting the system by some service curve characterization. There are some systems where we can produce results that are better than what their service-curve representation can provide.

Definition 2.11 (Packetizer P^L)**.** *A packetizer P^L is a causal, lossless, FIFO system that transforms a fluid, bit-by-bit stream into a packetized stream by releasing the packet's bits only when the last one is received.*

Appending a packetizer to a node does not increase the packet delay at this node [38, Theorem 1.7.1]. However, the packetizer increases the arrival curve of the departure flows and hence can increase the end-to-end delay bounds. In Chapter 6, we take into this in account when the arrival rate at the packetizer's input is limited.

2.1.3 End-to-End Worst-Case Delay Analysis for FIFO-per-Class Networks

So far, we considered a unique flow that crosses one or several systems. In a given time-sensitive network, it is unlikely that f be the only flow that crosses a system. The time-sensitive networks studied in the thesis are FIFO-per-class networks: We assume that flows are grouped into classes, packets inside one class are processed First-In-First-Out (FIFO), and classes are isolated using schedulers. We assume that flows are statistically assigned to a class. Also, we assume that flows are constrained by some arrival curves at their sources.

Now consider a flow f that belongs to class c. Packets of flows that belong to class c compete at each network node with flows of other classes; also, packets of different flows inside a class compete with each other within a class. In order to compute a bound on its end-to-end delays, the following steps are required:

1. Abstract the service offered by the system-level scheduling policy to class c at every network

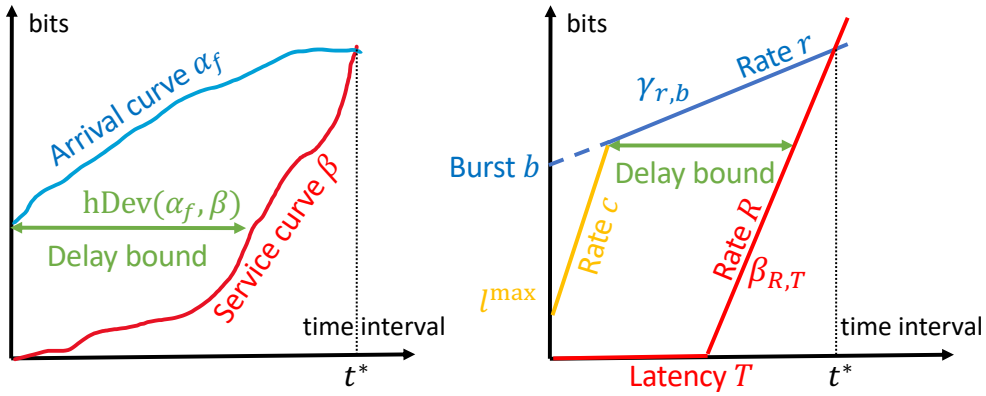


Figure 2.6: Left: Horizontal deviation of an arrival curve α_f and a service curve β, hDev(α_f, β), is a delay bound for a flow; a backlog bound can be obtained by finding the vertical deviation; these computations can be restricted to t^*, the first time after zero where α_f and β meet. Right: Illustration of a token-bucket arrival curve $\gamma_{r,b}$ and rate-latency strict service curve $\beta_{R,T}$; the yellow line shows the effect of line-shaping (also known as grouping) that improves the arrival curve, hence the delay bound.

node (also called residual service curve); this models the service offered to aggregate flows of class c at each node.

2. Worst-case analysis of the FIFO-per-class network of class c: Given the service characterization offered to class c at all network nodes and arrival curve constraints for flows of class c at their sources, obtain a bound for packets of flow f.

For the former, Per-class service curves have been vastly analyzed for some scheduling policies: Static Priority [17, Section 8.6.8.1] in [38, Section 6.2.1][54, Sections 7.3.2, 8.2.1]; Credit Based-Shaper (CBS) [17, Section 8.6.8.2] in [55, 56, 57, 58]; Time-Aware Shaper (TAS) [17, Sections 8.6.8.3-4, 8.6.9] in [54, Section 8.2.5] [59]. In Chapters 3 and 4, we also find such service curves for round-robin schedulers in this thesis.

2.1.3.1 Total Flow Analysis (TFA)

Total Flow Analysis (TFA) [62, 63, 64, 119] is a method to conduct worst-case analysis in a FIFO network. In a per-class network where a service curve is known for every class at every node, one instance of TFA is run per class, and it outputs per-node delay bounds as well as propagated burstiness for flows. TFA is simple, yet it can consider several important features, such as the effect of packetizer and line-shaping. If the network is feed-forward (i.e., cycle-free), for each node in a topological order, a delay bound and output burstiness bounds of flows are computed: the output burstiness bounds at a node are used as input by its successors in the induced graph. Else if the network has cyclic dependencies, no topological order can be defined, and a fixed point must be computed, using an iterative method [64]. If the iteration converges to a finite value for all delay and burstiness bounds, then the network is stable, and the computed bounds are valid. Otherwise, TFA diverges, and the network might be truly unstable or not. All versions of TFA (specifically, FP-TFA, SyncTFA, AsyncTFA, and AltTFA) are equivalent, i.e., they give the same bounds and stability regions [64].

Note that for networks with cyclic dependencies, all versions of TFA are designed and validated only

when arrival curves are token-bucket functions and service curves are rate-latency functions. In Chapter 6, we present and prove the validity of, a new version of TFA with arrival curves and service curves of generic shapes.

2.1.3.2 Single Flow Analysis (SFA)

SFA [62] computes an end-to-end delay bound for every flow of interest as follows and applies the PBOO principle by concatenating the residual service curves: First, it computes a residual service curve offered to the flow at every node in its path using [54, Theorem 7.5]. Second, it uses the concatenation results of Theorem 2.2 and obtains an end-to-end service curve offered to the flow of interest. Lastly, it uses the end-to-end service curve and the arrival curve of the flow at the source and applies Theorem 2.1 to obtain a delay bound. Unlike TFA, SFA benefits from PBOO, however, it does not account for the effect of line-shaping. Also, the first step, in the case of FIFO multiplexing, requires finding the optimal values for some parameters, which increases with the length of the flow path, thus increasing the complexity of SFA. SFA applies to feed-forward networks, and existing versions of SFA cannot be applied to networks with cyclic dependencies.

2.1.3.3 Pay Multiplexing Only Once (PMOO)

PMOO [62] goes in the same direction as SFA and computes an end-to-end service curve offered to the flow of interest by finding residual service curve at each node and concatenate them using Theorem 2.2; however, it groups interfering flows that share a sequence of systems, and then calculates an aggregate arrival-curve for each group. This results in accounting for the burst of interfering flows once within a flow path, which ultimately improves the end-to-end delay bound. PMOO can be only applied to FIFO tandem networks (where nodes are placed on a line and the path of each flow is a continuous sub-path of this line).

2.1.3.4 Least Upper Delay Bound (LUDB)

LUDB [22, 120, 121] relies on PMOO and is applied in FIFO tandem networks. It [120] provides tight delay bounds for FIFO sink-tree networks (where all flows leave the network at the same server). This process involves solving multiple Linear Programming (LP) problems, each providing an upper limit on the maximum delay. By solving each LP problem and choosing the smallest upper limit, the overall least upper bound can be determined. Authors in [22] extended the analysis of Least Upper Delay Bound (LUDB) for FIFO tandem networks to FIFO feed-forward networks; the method is called LUDB -FF.

2.1.3.5 Flow Prolongation

It has been observed that prolonging the paths of some interfering traffic might improve the bounds obtained by Pay Multiplexing Only Once (PMOO) and Least Upper Delay Bound (LUDB); the method is called flow prolongation [122]. Authors in [67] proposed a Graph neural Network (GNN) that predicts the best flow prolongations setting. We have implemented this GNN that is open source and publicly available in [123].

2.1.3.6 Mixed-Integer Linear Programming (MILP), Exponential-size Linear Programming (ELP), and Polynomial-size Linear Programming (PLP)

MILP [20] considers the arrival and departure time of a bit of interest; it then derives a number of time instants at every node, each of which is represented by a variable in MILP. Every time instant at a node is also associated with a variable that represents the value of the cumulative arrival function of the flows at this time instant. Network calculus relations such as arrival curve constraints, FIFO constraints, and service curve constraints are translated into linear constraints; the objective function to be maximized is the delay or backlog of the flow of interest. Also, it uses many integer variables. MILP provides tight delay bounds for FIFO feed-forward networks. However, it is applicable only to very small-sized networks.

ELP [20, 68] removes the integers variables of MILP, and thus provides less good bounds and loses the tightness property of MILP. However, the number of variables and constraints of ELP grows exponentially, and still, it is applicable only to very small-size FIFO feed-forward networks.

PLP [68] considerably makes ELP smaller by removing many variables and constraints. On the one hand, as the name suggests, the number of variables does not grow exponentially. On the other hand, PLP obtains less good bounds than ELP. To compensate for that, PLP uses delay bounds obtained by TFA and SFA as constraints; this improves the bounds obtained by PLP and guarantees that PLP always provides bounds that are less than or equal to those obtained by TFA or SFA. PLP can be applied to networks with cyclic dependencies, and it is known that it improves the stability region compared to TFA. We provide a more detailed background for PLP in Chapter 5.

2.1.4 Existing Worst-Case Delay Analysis Tools

The existing research or industrial network analysis tools related to network calculus are, but not limited to, DiscoDNC [66, 124], RTC Toolbox [125, 126], CyNC [127], RTaW-PEGASE [128], WoPANets [129], DelayLyzer [130], DEBORAH [131], NetCalBounds [132, 133], NCBounds [134], Siemens Network Planner (SINETPLAN) [135], xTFA [61], and Panco [136]. There also exist tools especially devoted to the min-plus algebra, such as Nancy [93], [137], and RTaW [92].

xTFA, DiscoDNC, and Panco support some of the methods we presented in the previous section. These tools altogether cover most of the widely recognizable methods within the community:

- **xTFA** is developed in Python and supports a more advanced TFA.
- **DiscoDNC** is developed in Java and partially uses linear programming with *CPLEX* [138] for LUDB.
- **Panco** is developed in Python and uses linear programming. So, it requires *lpsolve* [139] to execute TFA, SFA, PLP, and ELP.

In Appendix A, we present Saihu, a Python interface that integrates these three worst-case network analysis tools.

2.2 Lower Pseudo-Inverse

The lower pseudo-inverse $f^{\downarrow}$ of a function $f \in \mathcal{F}$ is defined by (see Fig. 2.7)

$$f^{\downarrow}(y) = \inf\{x | f(x) \geq y\} = \sup\{x | f(x) < y\} \tag{2.12}$$

We use the following property from [140, Sec. 10.1]:

$$\forall x, y \in \mathbb{R}^{+}, y \leq f(x) \Rightarrow x \geq f^{\downarrow}(y) \tag{2.13}$$

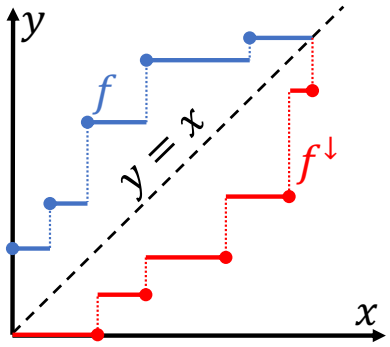


Figure 2.7: A function f and its lower pseudo inverse $f^{\downarrow}$; To obtain the lower pseudo-inverse of a function, first, the axes are flipped to find its inverse; if there are plateaux (i.e., horizontal lines) in f, they create vertical lines in its inverse; then, at each vertical line, the lowest value is taken.

2.3 Notation List Used Throughout the Thesis

Table 2.1: Notation List Used Throughout the Thesis

f	A flow
S	A system
α_f	An arrival curve for flow f at the source
β_n	A service curve offered by system S
A_f	Cumulative arrival function of flow f
D_f	Cumulative departure function of flow f
λ_c	Rate function with $\lambda_c(t) = ct$
$\beta_{R,L}$	Rate-latency function with $\beta_{c,L}(t) = \max(0, c(t-L))$
$\mathbb{R}^+$	Set of non-negative real numbers
$\mathscr{F}$	Set of wide-sense increasing functions $f : \mathbb{R}^+ \mapsto \mathbb{R}^+ \cup \{+\infty\}$
$\nu_{p,b}$	Stair function with $\nu_{p,b}(t) = b\left\lceil \frac{t}{p} \right\rceil$
$\gamma_{r,b}$	Token-bucket function with $\gamma_{r,b}(0) = 0$ and $\gamma_{r,b}(t) = rt + b$ for $t > 0$
$[x]^+$	$[x]^+ = \max(0, x)$
hDev	Horizontal deviation $\text{hDev}(\alpha, \beta) = \sup_{t \geq 0}\{\inf\{d \geq 0 \mid \alpha(t) \leq \beta(t+d)\}\}$
$f^{\downarrow}$	Lower pseudo inverse $f^{\downarrow} = \inf\{x \mid f(x) \geq y\} = \sup\{x \mid f(x) < y\}$
$\otimes$	Min-plus convolution $(f \otimes g)(t) = \inf_{0 \leq s \leq t}\{f(t-s) + g(s)\}$
$\oslash$	Min-plus deconvolution $(f \oslash g)(t) = \sup_{s \geq 0}\{f(t+s) - g(s)\}$
$[f]^+_{\uparrow}$	The non-decreasing and non-negative closure of f given by $[f]^+_{\uparrow}(t) = \sup_{s \leq t}[f(s)]^+$
vDev	Vertical deviation $\text{vDev}(\alpha, \beta) = \sup_{t \geq 0}\{\alpha(t) - \beta(t)\}$

Part II

Efficient and Accurate Worst-Case Delay Analysis of Time-Sensitive Networks with Round-Robin Schedulers

3 Strict Service Curves for Interleaved Weighted Round-Robin

In the realm of scheduling, where packets flow,
A dance of tasks, a rhythmic show.
Weighted Round-Robin, simple and clear,
Serves bursts of packets, without any fear.

But bursty service, a challenge we find,
Interleaved Round-Robin eases our mind.
Seeking bounds on delays, we delve deep,
Using network calculus, our promises to keep.

With IWRR, a strict curve we trace,
Derived from the pseudo-inverse's embrace.
Tight bounds we unveil, worst-case they be,
For constant-sized packets, a guarantee.

This service curve, dominant and grand,
Leaving WRR's curve far behind, unmanned.
Numerical examples, they come alive,
Revealing IWRR's reduction, as we strive.

So, let the chapter unfold, the findings astound,
As IWRR's triumph, in delays we confound.

Created with ChatGPT, free research preview (version May 24) [141]

Weighted Round-Robin (WRR) is a scheduling algorithm that is often used for scheduling tasks, or packets, in real-time systems or communication networks. The capacity is shared among several clients or queues by giving each of them a weight, which is a positive integer, and by providing more service to those with larger weights. Specifically, queues are visited one after the other, and when a queue i with weight w_i has an emission opportunity, it sends w_i packets, or less if fewer packets are present. The advantage of WRR is that it is fair and simple. However, the service is bursty because up to w_i packets can be served consecutively for queue i, which can cause a large worst-case waiting time for other queues. Interleaved Weighted Round-Robin (IWRR) mitigates this effect [84]. With IWRR, a queue i with weight w_i has w_i emission opportunities per round and can send up to one packet at every emission opportunity. In contrast, with WRR, it has one emission opportunity per round and can send up to w_i packets at every emission opportunity. Hence, IWRR spreads out emission opportunities

of each queue in a round, which is expected to result in a smoother service and lower worst-case delays. There exist several versions of IWRR; we focus on the simplest one, where queue i has emission opportunities in the first w_i cycles within a round (see Section 3.1 for a formal description of IWRR and Section 3.2 for WRR variants). As explained in Section 1.2.1, no prior literature exists for worst-case delay bounds with IWRR.

The network calculus approach was applied to WRR in [40, Sec. 8.2.4], where a *strict* service curve is obtained. As explained in Section 2.1.1.4, strict service curves are a special kind of service curve, hence can be used to derive valid delay bounds. Our first contribution is to obtain a strict service curve for IWRR. Compared to WRR, the interleaving in IWRR makes the analysis more difficult, and the method of proof in [40] cannot easily be extended. To circumvent this difficulty, we rely heavily on the method of pseudo-inverse, recalled in Section 2.2. As expected, the IWRR strict service curve dominates that of WRR, hence the resulting delay bounds for IWRR are always less than or equal to those for WRR.

The strict service curve enables us to obtain delay bounds by using network calculus, but such bounds might not always be tight, i.e., they might not always be equal to worst cases. This is because the strict service curve is an abstraction of the system. Our second contribution is to show that, for flows with packets of constant sizes, the strict service curve obtained for IWRR does provide tight delay bounds. We show that the same result holds for the existing strict service curve of WRR. It follows that, for flows with packets of constant size, the obtained characterization of IWRR (and of WRR) with a strict service is the best possible among all possible service curves, strict or not.

The strict service curve obtained for IWRR has some description complexity; see also Fig. 3.2. Therefore, we provide simplified lower bounds that can be used, at the expense of sub-optimality, when analytic, closed-form expressions are important.

The rest of this chapter is as follows: We describe our system model in Section 3.1. We describe the state-of-the-art in Section 3.2. In Section 3.3, we present our strict service curve for IWRR. In Section 3.4, we show that both the IWRR and WRR strict service curves are the best possible and that they give tight delay bounds for a with constant packet sizes. We use numerical examples to illustrate the worst-case latency improvement of IWRR over WRR obtained with our method in Section 3.5. We present proofs of results in Section 3.6. We conclude the chapter in Section 3.7. A summary of notation and symbols used in this chapter are given in Section 3.8.

3.1 System Model

We assume that we have $1, 2, \ldots, n$ classes and each flow is statistically assigned to a class: We consider a weighted round-robin subsystem that serves n input classes, has one queue per class, and uses a weighted round-robin algorithm to arbitrate between classes. The arbitration algorithm assumed in this chapter is IWRR, shown in Algorithm 3.1. When a packet of flow that belongs to class i enters the weighted round-robin subsystem, it is put into queue i. The weight of class i is w_i. IWRR runs an infinite loop of *rounds*. In one round, each queue i has w_i emission opportunities; one packet can be sent during one emission opportunity. The inner loop defines a *cycle*, where each queue is visited, but only those with a weight not smaller than the cycle number have an emission opportunity. The `send` instruction is assumed to be the only one with a non-null duration. Its actual duration depends on the packet size but also on the amount of service available to the entire weighted round-robin subsystem. See Fig. 3.1 for an illustration.

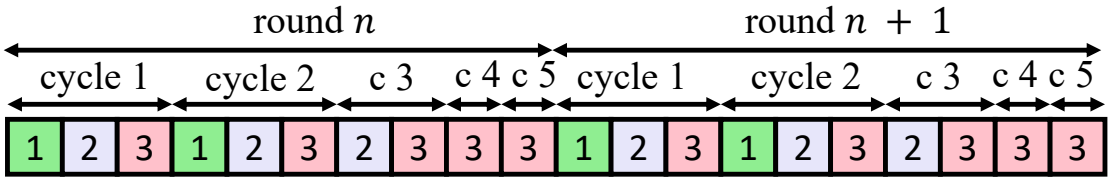


Figure 3.1: Emission opportunities on two successive rounds for IWRR with three classes and $w_1 = 2, w_2 = 3, w_3 = 5$. Mind that this is not temporal behavior: each opportunity can lead to an empty interval if the queue is empty at this time. Furthermore, the duration of each non-empty interval depends on the packet size and the aggregate service available (we do not assume constant rate service).

The weighted round-robin subsystem is itself placed in a larger system, and can compete with other queuing subsystems. For example, consider the case of a constant-rate server with several priority levels, without preemption, and where the weighted round-robin subsystem is at a priority level that is not the highest, as in [73, Sec. 8.6.8.3]. Assuming some arrival curve constraints for the higher priority traffic, the service received by the entire weighted round-robin subsystem can be modelled using a strict service curve [40, Sec. 8.3.2].

This motivates us to assume that the aggregate of all classes in the weighted round-robin subsystem receives a strict service curve, say $\beta \in \mathscr{F}$ that we call "aggregate strict service curve". If the weighted round-robin subsystem has exclusive access to a transmission line of rate c, then $\beta(t) = ct$ for $t \geq 0$. We assume that $\beta(t)$ is finite for every (finite) t and, without loss of generality, we assume β to be super-additive. Note that the aggregate strict service curve β, which models the service offered to the aggregate of all flows, should not be confused with the strict service curves such as β_i, which captures service offered to class i; strict service curves such as β_i are called "residual" strict service curves in some references.

Algorithm 3.1: Interleaved Weighted Round-Robin

Input: Integer weights $w_1 \leq w_2 \leq .. \leq w_n$

```
1  w_max ← max w_1,.., w_n;
2  while true do
       //A round starts.
3      for C ← 1 to w_max do
           //A cycle starts.
4          for i ← 1 to n do
5              if C ≤ w_i then
                   //Queue i is visited.
6                  if (not empty(i)) then
                       //A service for queue i.
7                      print(now, i);
8                      send(head(i));
9                      removeHead(i);
           //A cycle finishes.
       //A round finishes.
```

Here, we use the context of communication networks, but the results equally apply to real-time systems:

Simply map flow to task, packet to job, packet size to job execution time, and strict service curve to "delivery curve" [142, 143].

3.2 Related Works

One of the first use of round-robin scheduling in the network context appeared in [71], with a fairness objective, i.e., a fair way to share the bandwidth among sessions. It is also mentioned in [72] as a way to implement "fair queueing".

The term "Weighed Round-Robin" was coined in [84] as a generalization of round-robin to share the bandwidth "in proportion to prescripted weights" in the context of ATM (i.e., with constant-size packets). Two versions of the algorithm are presented in [84]. The former is presented in Algorithm 3.1: at cycle C (with C between 1 and $w_{\max}$), only classes with weight $w_i \geq C$ can emit one packet. We call this version IWRR. The latter version assumes that there exists for each class i a bit-list of length $w_{\max}$, $o_i \in \{0,1\}^{w_{\max}}$, such that $w_i = \sum_{k=1}^{w_{\max}} o_i[k]$. A class i can emit a packet at cycle C only if $o_i[C] = 1$. A strategy is given to build these vectors in [84] and is refined with fairness objectives in [144]. Call LIWRR (list-based IWRR) this version.

IWRR is modified into WRR/SB in [145] to enable some classes to send slightly more packets than permitted in a cycle, and to decrease accordingly at the next cycle.

As mentioned, plain WRR (which we simply call "WRR") enables each class i to send up to w_i packets every time it is selected [146]. A "Multiclass WRR" is also defined in [146]. Surprisingly, the authors of [146] were not aware of [84] and have re-invented LIWRR. Note that even if WRR was designed for packets of constant size, it has been applied in networks of variable-size packets such as Ethernet [73, Sec. 8.6, Sec. 8.6.8.3, Sec. 37], in request balancing in cloud infrastructures [74], in the LinuxVirtualServer scheduling [75], in network of chip [76], and so on. In fact, looking for expression "weighted round-robin" in the title or abstracts of papers index by Scopus returns more than 553 entries (May 2023), and Google references more than 4000 patents with this expression (May 2023). Unfortunately, when authors refer to WRR, they often do not explicit which version of WRR it is.

A WRR server is also a latency-rate server, with latency and rates given in [147] for packets of constant size. The latency result is generalized to LIWRR in [148]. Even if the notion of the latency-rate server is very close to the one of a service curve $\beta_{r,T}$ in network calculus, both notions are slightly different, and results cannot be directly imported from one theory to the other [149]. In [76], the authors consider a Network on Chip (NoC), with WRR arbitration at the flit level. A flit is the elementary data unit of the NoC, one flit is sent per CPU/NoC cycle. Assuming that the weights are such that packets are never fragmented by the arbiter, a strict service curve β_{R_i,T_i} for class i is found, with $R_i = \frac{w_i}{\sum_k w_k}$, $T_i = \sum_{j \neq i} w_j$.

WRR arbitration in an Ethernet switch is also considered in [150], with the assumption that all classes of an output port have the same constant packet size. It then computes, in the network calculus framework, a residual service with a service curve β_{R_i,T_i} with $R_i = \frac{w_i}{\sum_k w_k} C$, $T_i = \frac{\sum_{j \neq i} w_j}{C}$, where C is the link rate. We assume that the missing packet size in the T_i term was a typo. This network calculus result for conventional WRR arbitration in Ethernet is refined in [151], considering packets of variable size, leading to residual service with a strict service curve β_{R_i,T_i} with $R_i = \frac{w_i l_i^{\min}}{w_i l_i^{\min} + \sum_{j \neq i} w_j l_j^{\max}} C$ and $T_i = \frac{\sum_{j \neq i} w_j l_j^{\max}}{C}$ (cf. Eq. (1) and (2) in [151]) where $l_i^{\min}, l_i^{\max}$ are, respectively, lower and upper bounds on the size of the packets in class i. It refines this result by subtracting the part of the bandwidth not

used by interfering classes (considering their arrival curves).

Observe that computing a residual service with a $\beta_{R,T}$ curve is pessimistic as it assumes that, once the worst latency is paid, each packet is served with the long-term residual rate. Whereas, in reality, each packet, when it is selected for emission, is transmitted at full link speed up to completion. A residual service for the conventional WRR with a curve that is an alternation of full services and plateaus is given in [40, Sec. 8.2.4]. This effect of "full speed up to completion" can also be captured when computing the local delay of a server with $\beta_{R,T}$ service curve [152].

3.3 Strict Service Curves for IWRR

Our first result is a strict service curve for IWRR that, as we show in Section 3.4, is the best possible. We compare it to WRR and also give simpler, lower approximations.

Theorem 3.1 (Strict Service Curve of IWRR)**.** *Let S be a server shared by n classes that uses IWRR as explained in Section 3.1, with weight w_i for class i. Recall that the server offers a strict service curve β to the aggregate of the n classes. For any class i, l_i^{min} [resp. l_i^{max}] is a lower [resp. upper] bound on the packet size.*

Then, S offers to every class i a strict service curve β_i given by $\beta_i(t) = \gamma_i(\beta(t))$ with

$$\gamma_i = \lambda_1 \otimes U_i \tag{3.1}$$

$$U_i(x) \stackrel{def}{=} \sum_{k=0}^{w_i-1} \nu_{L_{tot},l_i^{min}}\left(\left[x - \psi_i(k l_i^{min})\right]^+\right) \tag{3.2}$$

$$L_{tot} = w_i l_i^{min} + \sum_{j,j\neq i} w_j l_j^{max} \tag{3.3}$$

$$\psi_i(x) \stackrel{def}{=} x + \sum_{j,j\neq i} \phi_{i,j}\left(\left\lfloor \frac{x}{l_i^{min}} \right\rfloor\right) l_j^{max} \tag{3.4}$$

$$\phi_{i,j}(x) \stackrel{def}{=} \left\lfloor \frac{x}{w_i} \right\rfloor w_j + \left[w_j - w_i\right]^+ + \min(x \bmod w_i + 1, w_j) \tag{3.5}$$

In the above, $\nu_{p,b}$ is the stair function, λ_1 is the unit rate function and $\otimes$ is the min-plus convolution, all are described in Fig. 2.3.

Furthermore, β_i is super-additive.

The proof is in Section 3.6.1. See Fig. 3.2 for some illustration of β_i. Our strict service curve captures the round-robin manner of IWRR: The fact that the service is interrupted in order to serve other queues, and then the class is served at a rate given by the server; each plateau (i.e., horizontal line) corresponds to a service interruption for class i. Observe that γ_i in Eq. (3.1) is the strict service curve obtained when the aggregate strict service curve is $\beta = \lambda_1$ (i.e., when the aggregate is served at a constant, unit rate). In the common case where β is equal to a rate-latency function, say $\beta_{c,T}$, we have $\beta_i(t) = \gamma_i(c(t-T))$ for $t \geq T$ and $\beta_i(t) = 0$ for $t \leq T$, namely, β_i is derived from γ_i by a rescaling of the x axis and a right-shift.

The key point of Theorem 3.1 is as follows: We take into account the details of IWRR in our analysis, thanks to the pseudo-inverse technique; all constraints are written without any attempts to invert them in closed form, and only at the final step they are inverted. Specifically, as shown in the proof, in $(s,t]$, a backlogged period (see Definition 2.9) of the class of interest i, the service received by an

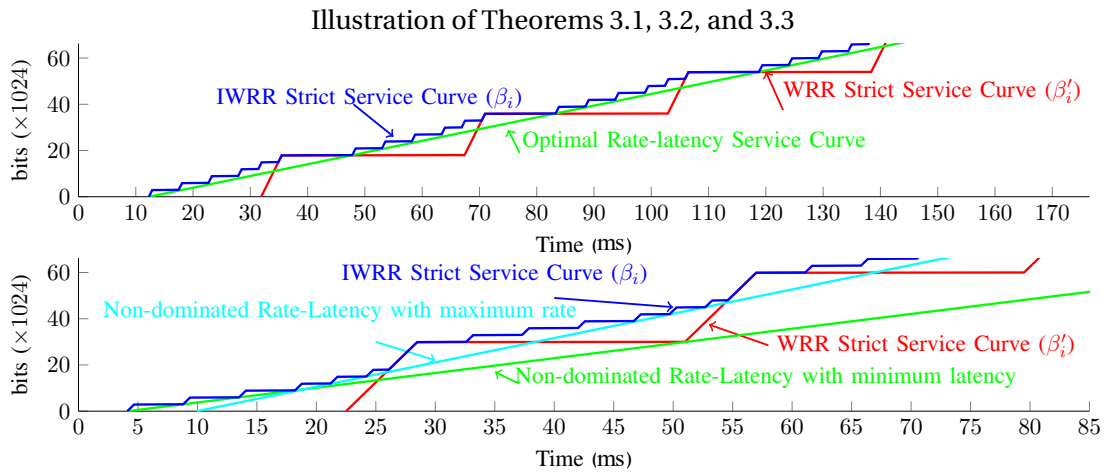


Figure 3.2: Strict service curves obtained in Section 3.3 for an example with four input classes, weights $= \{4,6,7,10\}$, $l^{\min} = \{4096,3072,4608,3072\}$ bits, $l^{\max} = \{8704,5632,6656,8192\}$ bits and $\beta(t) = ct$ with $c = 10$ Mb/s (i.e., the aggregate of all flows is served at a constant rate). The figure shows the IWRR service curve β_i and the WRR strict service curve β'_i for two of the classes; it also shows the non-dominated rate-latency strict service curves $\beta_{r_0^*,T_0^*}$ and $\beta_{r_{k^*}^*,T_{k^*}^*}$ of Theorem 3.3 (in the top panel both are equal).

interfering class j (i.e., $D_j(t) - D_j(s)$) is upper bounded as a function of the service received by class i (i.e., $\phi_{i,j}(D_i(t) - D_i(s))$, where $\phi_{i,j}$ is defined in (3.5).

As mentioned in Section 6.1, any strict service curve that is not super-additive can be improved, by replacing it with its super-additive closure. The last statement in the theorem guarantees that it is not possible to improve the obtained service curve in this way.

We now compare to WRR. The best known service curve for (non-interleaved) WRR is given in [40, Sec. 8.2.4] and is

$$\beta'_i(t) = (\lambda_1 \otimes \nu_{L_{\text{tot}},q_i})\left(\left[\beta(t) - Q_i\right]^+\right) \tag{3.6}$$

with $q_i = w_i l_i^{\min}$ and $Q_i = \sum_{j,j\neq i} w_j l_j^{\max}$. In Section 3.4, we show that $\beta'_i(t)$ is indeed the best possible strict service curve for WRR. Furthermore, it is dominated by the strict service curve for IWRR:

Theorem 3.2. *With the assumptions in Theorem 3.1 and in Eq.* (3.6)*:*

$$\beta'_i \leq \beta_i \tag{3.7}$$

The proof is in Section 3.6.2. Figure 3.2 illustrates how the strict service curve for IWRR improves on that for WRR, by providing a smoother, and generally larger, service.

The service curve found in Theorem 3.1 is the best possible one but has a complex expression. If there is interest in a simpler expression, any lower bounding function is a strict service curve; in particular, the strict service curve β'_i for WRR is also a valid, though sub-optimal, strict service curve for IWRR. There is often interest in service curves that are rate-latency functions. Observe that, if the aggregate service curve β is a rate-latency function, then replacing γ_i by a rate-latency lower-bounding function also yields a rate-latency function for β_i, and vice-versa. Therefore, we are interested in rate-latency functions that lower bound γ_i.

Among all of these, there is not a single best one, as some have a smaller latency while others have a larger rate. We say that a rate-latency function $\beta_{r,T}$ that lower bounds γ_i is non-dominated if there is no other rate latency function $\beta_{r',T'}$ that lower bounds γ_i and dominates $\beta_{r,T}$, i.e., such that $r' \geq r$ and $T' \leq T$. The following result gives all such non-dominated rate-latency functions. Let $r^* = \frac{q_i}{L_{\text{tot}}} = \frac{w_i l_i^{\min}}{L_{\text{tot}}}$, $r_{w_i-1} = 1$, and

$$r_k = \frac{l_i^{\min}}{\psi_i((k+1)l_i^{\min}) - \psi_i(kl_i^{\min})},\ 0 \leq k < w_i - 1 \tag{3.8}$$

$$k^* = \min\{0 \leq k < w_i \mid r_k \geq r^*\} \tag{3.9}$$

$$r_k^* = \min(r_k, r^*),\ 0 \leq k \leq k^* \tag{3.10}$$

Theorem 3.3. *With the assumptions in Theorem 3.1 and the definitions* (3.8)-(3.10)*, a rate-latency function $\beta_{r,T}$ lower bounds γ_i and is non-dominated if and only if $r = r_{k^*}^*$ and $T = \psi_i(k^* l_i^{min}) - \frac{k^* l_i^{min}}{r}$, or $r_{k-1}^* \leq r < r_k^*$ and $T = \psi_i(kl_i^{min}) - \frac{kl_i^{min}}{r}$ for some integer k with $0 < k \leq k^*$. Among all such rate-latency functions, the one with the lowest latency is $\beta_{r_0^*,T_0^*}$ and the one with the largest rate is $\beta_{r_{k^*}^*,T_{k^*}^*}$.*

The proof is in Section 3.6.3. Figure 3.2 illustrates $\beta_{r_0^*,T_0^*}$ and $\beta_{r_{k^*}^*,T_{k^*}^*}$ in some examples. Observe that $k \mapsto r_k^*$ is wide-sense increasing with k for $0 \leq k \leq k^*$, but the values of r_k^* are not necessarily all distinct. It can also occur that $k^* = 0$ (as in the top panel of Fig. 3.2); in which case, there is one optimal rate-latency service curve. In general, however, this does not occur, and a simple lower bounding approximation can be obtained with the supremum of all non-dominated rate-latency service curves, as given by the next theorem.

Theorem 3.4. *With the assumptions in Theorem 3.3, the supremum of all non-dominated rate-latencies is equal to* $\max\left(\beta_{r_0^*,T_0^*}, \ldots, \beta_{r_{k^*}^*,T_{k^*}^*}\right)$*, and it is the largest convex function that lower bounds γ_i.*

The proof is in Section 3.6.4. There is often interest in service curves that are piecewise linear and convex. Specifically, convex piecewise-linear functions are stable under addition and maximum, and the min-plus convolution can be computed in automatic tools very efficiently [40, Sec. 4.2]. The above theorem thus gives the best such strict service curve.

3.4 Tightness

We first show that the strict service curve we have obtained is the best possible. The proofs of all results in this section are in Section 3.6.

3.4.1 Tightness of Strict Service Curve

Theorem 3.5. *(Tightness of the IWRR Service Curve) Consider a weighted round-robin subsystem that uses the IWRR scheduling algorithm, as defined in Section 3.1. Assume the following system parameters are fixed: the number of input classes, the weight w_j allocated to every class j, the bounds on packet sizes l_j^{min} and l_j^{max} for every class j, and the strict service curve β for the aggregate of all flows. We assume that β is Lipschitz-continuous, i.e., there exists a constant $K > 0$ such that $\frac{\beta(t)-\beta(s)}{t-s} \leq K$ for all $0 \leq s < t$. Let i be the index of one of the classes.*

Assume that $b_i \in \mathscr{F}$ is a strict service curve for class i in any system that satisfies the specifications above. Then $b_i \leq \beta_i$ where β_i is given in Theorem 3.1.

The proof is in Section 3.6.5. The idea of the proof is as follows: For any value of the system parameters, for any $\tau > 0$, and for any class i, we create a trajectory scenario of a system such that

$$\begin{aligned} &\exists s \geq 0, (s, s+\tau] \text{ is backlogged for class } i \\ &\text{and } D_i(s+\tau) - D_i(s) = \beta_i(\tau) \end{aligned} \tag{3.11}$$

,i.e., for the class of interest i, we create a backlogged period (see definition 2.9) of duration τ where the service received by class i is exactly equal to $\beta_i(\tau)$. Then, it follows that every other strict service curve b_i is upper bounded by β_i. Note that assuming the aggregate strict service curve β is Lipschitz-continuous does not appear to be a restriction as the rate at which data is served has a physical limit. Interestingly, we obtain a similar result for WRR. Recall that β'_i is the strict service curve for class i, described in Eq. (3.6), which was obtained in [40, Sec. 8.2.4].

Theorem 3.6. *(Tightness of the WRR Service Curve) Theorem 3.5 is also valid if we replace IWRR with WRR. Specifically, using WRR as a scheduling policy, β'_i is the largest possible strict service curve for class i.*

3.4.2 Tightness of Delay Bounds with Constant Packet Sizes

Having obtained the best-possible strict service curve does not guarantee that the delay bounds derived from it are tight, i.e., are worst-case delays. This is because a service curve is only an abstraction of the system; and we have obtained a strict service curve, and non-strict service curves might provide better results. However, we show that, for flows of packets of constant size, we do obtain tight delay bounds. We show that it holds for IWRR and for WRR.

Recall that a delay bound requires the knowledge of an arrival curve α_i for the class of interest. If flows of this class generate only packets of length l, then α_i can be assumed to be a multiple of l and sub-additive. A delay bound for this class is then equal to $\text{hDev}(\alpha_i, \beta_i)$ (see Theorem 2.1).

Theorem 3.7. *(Tightness of Delay Bound for IWRR with Constant Packet Size) Consider a system, as in Theorem 3.5, with the additional assumption that, for the class of interest i, $l_i^{min} = l_i^{max} = l$.*

Let $\alpha_i \in \mathscr{F}$ be a sub-additive function that is an integer multiple of l, and assume that class i has α_i as an arrival curve. The network calculus delay bound is tight, i.e, there exists a trajectory where the delay of one packet of class i is equal to $hDev(\alpha_i, \beta_i)$.

Theorem 3.8. *(Tightness of Delay Bound for WRR with Constant Packet Size) Theorem 3.7 is also valid for the WRR policy.*

3.5 Numerical Examples

To compare IWRR and WRR worst-case delays, we provide some numerical examples. First, we consider a system of 8 input classes $f_1, \ldots, f_8$ with respective weights $\{22, 27, 28, 30, 30, 34, 41, 45\}$ and $l^{\min} = l^{\max} = l = 7119$ bit. Let the aggregate service, β, be a constant bit rate of 10 Mb/s. For every class i, we compute the IWRR and WRR strict service curves β_i, β'_i. Then, for every i, we generate $N = 1000$ token-bucket

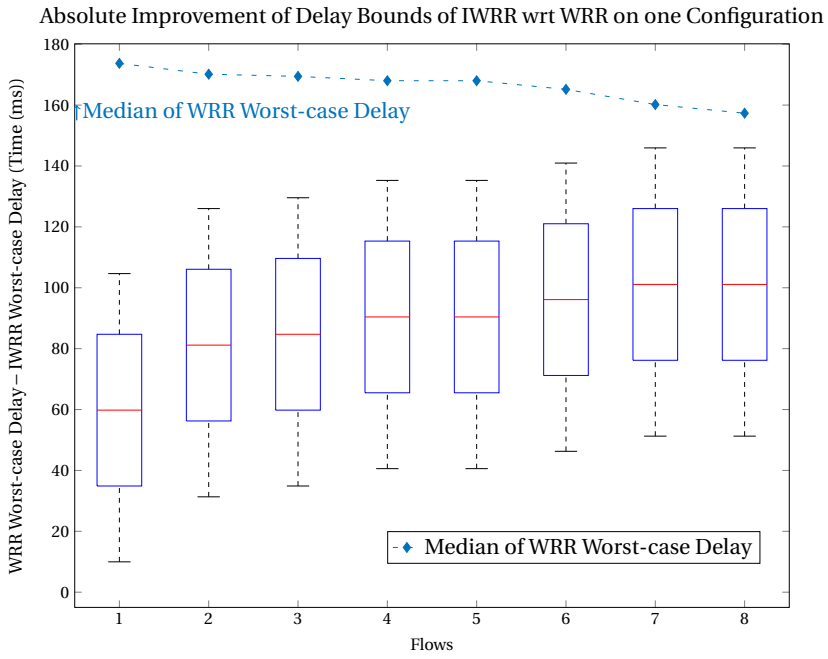


Figure 3.3: Box-and-whisker plots of difference between WRR and IWRR delay bounds with weights $\{22,27,28,30,30,34,41,45\}$ and $l = 7119$ bit with random arrival curves. Median WRR delay bounds are also provided.

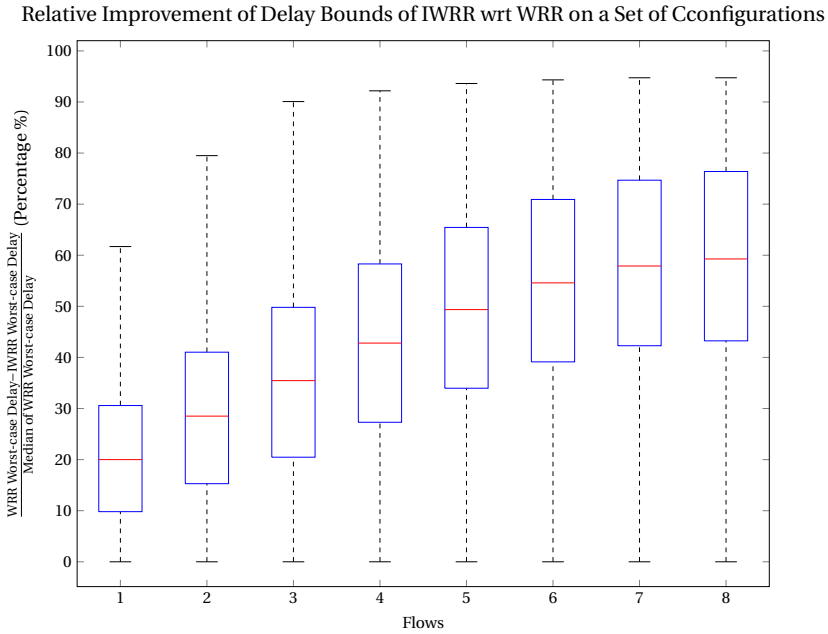


Figure 3.4: Box-and-whisker plots of difference between WRR and IWRR delay bounds normalized to the median of WRR delay bounds, for several systems with weights picked uniformly at random in $[10,50]$, assigned to a class by increasing order, and a packet length picked uniformly at random in $[64,1522]$ bytes.

arrival curves γ_{r,b_k}, $k = 1,\ldots,N$, with rate $r = 0.5$ Mb/s and burst b_k picked uniformly at random in $[1,20]$ packets. Then, we use $\alpha_i^k = \lceil \frac{\gamma_{r,b_k}}{l} \rceil l$ to satisfy the conditions of Theorems 3.7 and 3.8 and to compute $d_i^k = h(\alpha_i^k, \beta_i)$ and $\dot{d}_i^k = h(\alpha_i^k, \beta_i')$. Figure 3.3 gives the box-and-whisker plots of the $\dot{d}_i^k - d_i^k$ series. The median of WRR delay bounds $\dot{d}_i^k$ are also provided to illustrate the improvement.

Second, we repeated the same study for $M = 10000$ sets of system parameters. For each system, we choose the weights of 8 classes by picking them uniformly at random between 10 and 50, and we pick a packet length l uniformly at random between 64 to 1522 bytes. For each experiment, we call class 1 the class with the smallest weight, class 2 with the second smallest weight, and so on. As the scale of delay bounds depends on the choices of weights and the packet length, the $\dot{d}_i^k - d_i^k$ series are divided by $\dot{d}_i^{\tilde{m}}$, the median of WRR delay bounds for class i. Figure 3.4 gives the box-and-whisker plots of the $\frac{\dot{d}_i^k - d_i^k}{\dot{d}_i^{\tilde{m}}}$ series. Using IWRR improves worst-case delays, as expected, and the improvement is larger for classes with larger weights.

3.6 Proofs

3.6.1 Proof of Theorem 3.1

The idea of proof is as follows. We consider a backlogged period (see definition 2.9) $(s,t]$ of the class of interest i, and we let p be the number of packets of class i that are entirely served during this period. For every other class j, the number of packets that are entirely served is upper bounded by a function of p, given in Lemma 3.3. Also, p is upper bounded by a function of the amount of service received by class i in Lemma 3.5. Combining these two results gives an implicit inequality for the total amount of service in Eq. (3.24). By using the technique of pseudo-inverse, this inequality is inverted and provides a lower bound for the amount of service received by the class of interest.

3.6.1.1 Key Variables and Basic Properties

Let $(s,t]$ be a backlogged period of class i. Let (τ_k, f_k) be couples of (instant, class), printed at line 7 of Algorithm 3.1. Note that $\tau_k < \tau_{k+1}$ as the `send` instruction has a non-null duration (because the aggregate service curve β is Lipschitz continuous). Let $\sigma(0), \sigma(1), \ldots$ be the sequence of service opportunities for class i at or after s, i.e., $\sigma(0) = \min\{m \mid \tau_m \geq s, f_m = i\}$ and $\sigma(k) = \min\{m \mid \tau_m > \tau_{\sigma(k-1)}, f_m = i\}$. The kth service opportunity for class i occurs at time $\tau_{\sigma(k-1)}$; we say that it is "complete" if $\tau_{\sigma(k-1)+1} \leq t$, i.e., the interval taken by this service is entirely in $[s,t]$. Let $p \geq 0$ be the number of complete service opportunities. Observe that it is possible that $p = 0$, and it might happen that $\tau_{\sigma(p)} < t$ or $\tau_{\sigma(p)} \geq t$ (see Fig. 3.5).

In each service of class i, during a backlogged period, it sends one packet with a length $\geq l_i^{\min}$, thus, for all $k = 0 \ldots (p-1)$, we have $D_i(\tau_{\sigma(k+1)}) - D_i(\tau_{\sigma(k)}) \geq l_i^{\min}$, therefore

$$D_i(\tau_{\sigma(p)}) - D_i(\tau_{\sigma(0)}) \geq p l_i^{\min} \tag{3.12}$$

3.6.1.2 Amount of Service to Other classes

In order to upper bound the number of emission opportunities for another class j, we first find an expression, in Lemma 3.1, for the number of emission opportunities for class j between two consecutive

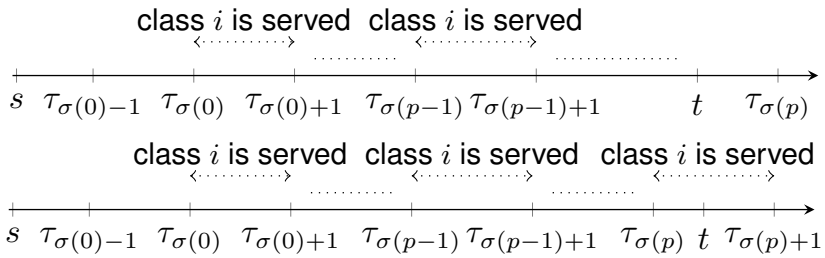


Figure 3.5: Illustration of two possible cases of $\tau_{\sigma(p)} \geq t$ and $\tau_{\sigma(p)} < t$.

emission opportunities for class i. Lemma 3.2 then finds an upper bound on the number of emission opportunities for class j in $(s, \tau_{\sigma(p)})$, as a function of the cycle number (variable C in Algorithm 3.1) at $\tau_{\sigma(0)}$. Lastly, Lemma 3.3 maximizes the previous upper bound over all values of C.

Lemma 3.1. *The number of emission opportunities for class $j \neq i$ between two consecutive emission opportunities for class i, given that the latter emission opportunity for class i occurs at cycle C, is equal to*

$$q_{g,f}(C) = \begin{cases} 0 & \text{if } 1 < C \leq w_i \text{ and } w_j < C \\ 1 & \text{if } 1 < C \leq w_i \text{ and } w_j \geq C \\ \left[w_j - w_i\right]^+ + 1 & \text{if } C = 1 \end{cases} \tag{3.13}$$

Proof: According to Algorithm 3.1, class i has emission opportunities only in the first w_i cycles of each round. Both emission opportunities are either in the same round (Case 1) or in two consecutive rounds (Case 2). As C is the cycle number for the second emission opportunity for class i, Case 1 can occur only when $1 < C \leq w_i$, and Case 2 can occur when $C = 1$. For Case 1, we further differentiate between $w_j < C$ and $w_j \geq C$.

Case 1a: $1 < C \leq w_i$ and $w_j < C$: Queue j does not have an emission opportunity in cycle C because $w_j < C$. Also, we must have $w_j < w_i$, thus queue j does not have any emission opportunity after i in cycle $C-1$. Hence, $q_{i,j}(C) = 0$.

Case 1b: $1 < C \leq w_i$ and $w_j \geq C$: If $w_j > w_i$, then queue j has an emission opportunity after queue i in cycle $C-1$. If $w_j = w_i$, then queue j has an emission opportunity before i in cycle C, or after i in cycle $C-1$. Else, $C \leq w_j < w_i$ and queue j has an emission opportunity in cycle C, before i. In all cases, $q_{i,j}(C) = 1$.

Case 2: $C = 1$: The first emission opportunity for i is in the last cycle of a round that includes i (cycle w_i). If $w_j > w_i$, then queue j has an emission opportunity in the rest of cycle w_i and also has emission opportunities during the next $(w_j - w_i)$ cycles of the last round. In this case, $q_{i,j}(C) = w_j - w_i + 1$, which is also the value in the last line of Eq. (3.13). Else if $w_j = w_i$, queue j has an emission opportunity before i in this cycle or after i in cycle w_i of the first round, thus $q_{i,j}(C) = 1$, which is also the value in the last line of Eq. (3.13). Else, $w_j < w_i$ and queue j has an emission opportunity before i in this cycle. Here too, $q_{i,j}(C) = 1$, the value in the last line of Eq. (3.13). □

Lemma 3.2. *The number of emission opportunities for class $j \neq i$ in $(s, \tau_{\sigma(p)})$, for any backlogged period $(s, t]$ of class i with p complete services, given that the first service starts at cycle number C (cycle number*

at time $\tau_{\sigma(0)}$) is upper bounded by

$$q'_{i,j}(C,p) \stackrel{def}{=} \sum_{k=0}^{p} q_{i,j}\left((C+k-1) \mod w_i + 1\right) \tag{3.14}$$

Also, let $C'(p)$ be the cycle number at $\tau_{\sigma(p)}$. Then,

$$C'(p) = \left(C+p-1\right) \mod w_i + 1 \tag{3.15}$$

Proof: By induction on p.

Base Case: $p = 0$

In this case, $q'_{i,j}(C,0)$ is the number of emission opportunities for class j between two consecutive emission opportunities for class i that by Lemma 3.1, is equal to $q_{i,j}(C)$. As $1 \le C \le w_i$, $(C-1) \bmod w_i + 1 = C$ thus $q_{i,j}(C) = q_{i,j}\left((C-1) \bmod w_i + 1\right)$. This shows Eq. (3.14). Also, by definition, $C'(0) = C$; using again $(C-1) \bmod w_i + 1 = C$ shows that Eq. (3.15) holds.

Induction step:

We assume that Eq. (3.14) and Eq. (3.15) hold for $p-1$, and we want to show that they also hold for p.

First, let us prove Eq. (3.15). There are two possible cases: (a) if $0 \le C'(p-1) < w_i$, then both $(p-1)$st and pth emission opportunities occur in the same round, thus $C'(p) = C'(p-1) + 1$. By the induction hypothesis, $\left(C+p-2\right) \bmod w_i + 1 < w_i$, i.e., $\left(C+p-2\right) \bmod w_i < w_i - 1$. Note that, for any integer x

$$(x+1) \mod w = \begin{cases} (x \mod w) + 1 \text{ if } (x \mod w) < w-1 \\ 0 \text{ otherwise} \end{cases} \tag{3.16}$$

By using Eq. (3.16), we obtain that $C'(p)$ is given by Eq. (3.15) as required. (b) In the second case, $C'(p-1) = w_i$, then the next emission opportunity occurs in the first cycle of the next round, thus $C'(p) = 1$. Here too, applying Eq. (3.16) shows that $C'(p)$ is given by Eq. (3.15) as required.

Then, we prove Eq. (3.14). Let N be the number of emission opportunities for class j in $[s, \tau_{\sigma(p)})$. N is the sum of N_1, the number of emission opportunities in $[s, \tau_{\sigma(p-1)})$, and N_2, the number of emission opportunities in $(\tau_{\sigma(p-1)}, \tau_{\sigma(p)})$. By the induction hypothesis, $N_1 \le q'_{i,j}\left(C, p-1\right)$. Also, by Lemma 3.1, we have $N_2 \le q_{i,j}(C'(p))$. Thus, by using Eq. (3.15) which was just shown to also hold for p, we obtain

$$N \le \sum_{k=0}^{p-1} q_{i,j}\left((C+k-1) \mod w_i + 1\right) + q_{i,j}\left(\left(C+p-1\right) \mod w_i + 1\right) \tag{3.17}$$

where the right-hand side is equal to $q'_{i,j}(C,p)$ as required. □

Lemma 3.3. *For any backlogged period $(s,t]$ of class i with p complete services, the number of emission opportunities for class $j \neq i$ in $(s, \tau_{\sigma(p)})$ is upper bounded by $\phi_{i,j}(p)$, defined in Eq.* (3.5).

Proof: Lemma 3.2 gives the number of emission opportunities for class $j \neq i$ in $(s, \tau_{\sigma(p)})$, for any backlogged period $(s,t]$ of class i with p complete services, when the first service starts at cycle number C (cycle number at time $\tau_{\sigma(0)}$). To obtain the lemma, we maximize this result over C. We show the following properties.

(P1) For any integer $C \in [1, w_i]$,

$$\sum_{k=0}^{w_i-1} q_{i,j}\left((C+k-1) \mod w_i+1\right) = w_j \tag{3.18}$$

The mapping $k \mapsto (C+k-1) \bmod w_i + 1$ is one-to-one from $\{0,...,w_i-1\}$ onto $\{1,...,w_i\}$, thus the left-hand side of Eq. (3.18) is equal to $\sum_{k=1}^{w_i} q_{i,j}(k)$ that as we show now, is equal to w_j. First, we have $q_{i,j}(1) = \left[w_j - w_i\right]^+ + 1$. Also, $q_{i,j}(k) = 1$ when $k > 1$ and $w_j \geq k+1$. Thus, $\sum_{k=2}^{w_i} q_{i,j}(k) = \min(w_i - 1, w_j - 1)$ and finally the left-hand side is equal to $\left[w_j - w_i\right]^+ + \min(w_i - 1, w_j - 1) + 1$, which is equal to w_j.

(P2) For any integers $C \in [1, w_i]$ and $p \geq 0$,

$$q'_{i,j}\left(C, p\right) = \left\lfloor \frac{p}{w_i} \right\rfloor w_j + \sum_{k=0}^{p \bmod w_i} q_{i,j}\left((C+k-1) \mod w_i+1\right) \tag{3.19}$$

$q_{i,j}$ is a periodic function with period w_i. By (P1), the sum over one complete period is w_j. Also, we can write $p = \left\lfloor \frac{p}{w_i} \right\rfloor w_i + p \bmod w_i$. Thus, we have $\left\lfloor \frac{p}{w_i} \right\rfloor$ complete rounds, and the sum in Eq. (3.19) is the remainder.

(P3) $q_{i,j}$ is a wide-sense decreasing function. This means that for any integer $k \in [1, w_i)$, $q_{i,j}(k+1) \leq q_{i,j}(k)$. If $k = 1$, this follows from $q_{i,j}(1) \geq 1$ and $q_{i,j}(2) \leq 1$. Else if $k \leq w_j < k+1$, then $q_{i,j}(k+1) = 0$ and $q_{i,j}(k) = 1$. Else, they are equal. Hence, in all cases, the property holds.

(P4) For any integer $C \in [1, w_i]$ and $p \geq 0$,

$$q'_{i,j}\left(C, p\right) \leq q'_{i,j}\left(1, p\right) \tag{3.20}$$

By using (P2), we should show that

$$\sum_{k=0}^{p \bmod w_i} q_{i,j}\left((C+k-1) \mod w_i+1\right)$$

is upper bounded by $\sum_{k=0}^{p \bmod w_i} q_{i,j}\left(k \bmod w_i+1\right)$. Note that here we have $k \bmod w_i = k$. Both sides are the sum of $a \stackrel{def}{=} p \bmod w_i + 1$ unique elements of the set $\{q_{i,j}(k)\}_{k\in[1,w_i]}$. By (P3), the right-hand side is the maximum sum of a unique elements of this set.

(P5) For any integer $p \geq 0$,

$$q'_{i,j}\left(1, p\right) = \phi_{i,j}(p) \tag{3.21}$$

We apply (P2) with $C = 1$ to compute $q'_{i,j}\left(1, p\right)$. Then, the sum in the right-hand side of Eq. (3.19) is equal to $\sum_{k=0}^{p \bmod w_i} q_{i,j}(k+1)$, as $k \bmod w_i = k$. Then, by using the same argument after Eq. (3.18), it is equal to $\left[w_j - w_i\right]^+ + 1 + \min(p \bmod w_i, w_j - 1)$, which, by Eq. (3.5), is precisely $\phi_{i,j}(p)$.

The lemma then follows directly from (P4) and (P5). □

Lemma 3.4. *For every class $j \neq i$,*

$$D_j(t) \leq D_j(\tau_{\sigma(p)}) \tag{3.22}$$

Proof: If $t \leq \tau_{\sigma(p)}$, the result follows from D_j being wide-sense increasing. Else, we have $t > \tau_{\sigma(p)}$; this

implies that class i is served during $[\tau_{\sigma(p)}, t]$; thus for any other class j, $D_j^*(t) = D_j^*(\tau_{\sigma(p)})$. □

3.6.1.3 Amount of Service to Class of Interest

Lemma 3.5. *The number of complete services, p, of class of interest, i, in $(s,t]$ is upper bounded by*

$$p \leq \left\lfloor \frac{D_i(t) - D_i(s)}{l_i^{min}} \right\rfloor \tag{3.23}$$

Proof: First, $D_i(s) \leq D_i(\tau_{\sigma(0)})$, as $s \leq \tau_{\sigma(0)}$ and D_i is wide-sense increasing. Second, consider the two cases in 3.6.1.1. If $t \geq \tau_{\sigma(p)}$, the property holds. Else, the scheduler in not serving class i in $[\tau_{\sigma(p-1)+1}, \tau_{\sigma(p)})$, thus, $D_i(t) = D_i(\tau_{\sigma(p)})$. Hence, in both cases $D_i(t) \geq D_i(\tau_{\sigma(p)})$. By Eq. (3.12), $D_i(t) - D_i(s) \geq p l_i^{\min}$. Then, observe that p is an integer. □

3.6.1.4 Total Amount of Service

Lemma 3.6. *For any backlogged period $(s,t]$ of the class of interest i,*

$$\beta(t-s) \leq \psi_i\left(D_i(t) - D_i(s)\right) \tag{3.24}$$

where ψ_i is defined in Eq. (3.4).

Proof: As the interval $(s,t]$ is a backlogged period, by the definition of the strict service curve for the aggregate of classes, $\beta(t-s) \leq \sum_j D_j(t) - D_j(s)$. We upper bound $D_j(t)$ for all $j \neq i$ by applying Lemma 3.4,

$$\beta(t-s) \leq (D_i(t) - D_i(s)) + \sum_{j, j\neq i} D_j(\tau_{\sigma(p)}) - D_j(s) \tag{3.25}$$

Each class j has at most $\phi_{i,j}(p)$ emission opportunities during $\left(s, \tau_{\sigma(p)}\right)$ (Lemma 3.3) and can send at most one packet of maximum size in each. Thus,

$$\beta(t-s) \leq (D_i(t) - D_i(s)) + \sum_{j, j\neq i} \phi_{i,j}(p)\, l_j^{\max} \tag{3.26}$$

Also, Lemma 3.5 finds an upper bound on p. Thereby,

$$\beta(t-s) \leq (D_i(t) - D_i(s)) + \sum_{j, j\neq i} \phi_{i,j}\left(\left\lfloor \frac{D_i(t) - D_i(s)}{l_i^{\min}} \right\rfloor\right) l_j^{\max} \tag{3.27}$$

where the right-hand side is equal to $\psi_i(D_i(t) - D_i(s))$. □

3.6.1.5 Lower Pseudo-inverse of ψ_i

Our next step is to invert Eq. (3.24) by computing the lower-pseudo inverse of ψ_i. As the calculus of pseudo inverses applies to wide-sense increasing functions, we first show:

Lemma 3.7. *ψ_i, defined in Eq.* (3.4)*, is wide-sense increasing.*

Proof: It is sufficient to show that $\phi_{i,j}$, defined in Eq. (3.5), is a wide-sense increasing function. For

any non-negative integers x and y such that $y \le x$, we can write $x = k w_i + (x \bmod w_i)$ and $y = k' w_i + (y \bmod w_i)$, where k and k' are non-negative integers. We must have $k \le k'$. If $k = k'$, we know that $(y \bmod w_i \le x \bmod w_i)$ and $\left\lfloor \frac{x}{w_i} \right\rfloor = \left\lfloor \frac{y}{w_i} \right\rfloor$. Hence, $\phi_{i,j}(y) \le \phi_{i,j}(x)$. Else, $k > k'$ and $\left\lfloor \frac{x}{w_i} \right\rfloor > \left\lfloor \frac{y}{w_i} \right\rfloor$. Thereby, $\phi_{i,j}(x)$ is at least one w_j larger than $\phi_{i,j}(y)$. Hence, $\phi_{i,j}(y) < \phi_{i,j}(x)$. □

Lemma 3.8. *Let $g_0, g_1, \ldots, g_k, \ldots$ be a non-negative sequence such that $g_{k+1} - g_k \ge 1$. The sequence can be extended to a function in $\mathscr{F}$ by $g(x) = g_{\lfloor x \rfloor}$ and let $g^{\downarrow}$ be its lower pseudo-inverse, so that $g^{\downarrow}(y) = k+1 \in \mathbb{N} \Leftrightarrow g_k < y \le g_{k+1}$. Define $f \in \mathscr{F}$ by $f(x) = g_{\lfloor x \rfloor} + x \bmod 1$. Then, $f^{\downarrow} = \lambda_1 \otimes g^{\downarrow}$.*

Proof: Observe that convolving $g^{\downarrow}$ with λ_1 consists in smoothing the unit steps with a slope of 1 (Fig. 2.3). Thus $(\lambda_1 \otimes g^{\downarrow})(y) = k + y - g_k$ whenever $g_k \le y \le g_k + 1$ and $(\lambda_1 \otimes g^{\downarrow})(y) = k+1$ whenever $g_k + 1 \le y \le g_{k+1}$.

Also, f is piecewise linear and can be inverted in a closed form on every interval where it is linear. A direct calculation gives $f^{\downarrow}(y) = k + y - g_k$ whenever $g_k \le y \le g_k + 1$ and $f^{\downarrow}(y) = k+1$ whenever $g_k + 1 \le y \le g_{k+1}$. □

Lemma 3.9. *Let $f \in \mathscr{F}$ and $l, m > 0$. Define $h \in \mathscr{F}$ by $h(x) = m f\left(\frac{x}{l}\right)$. Then, for all $y \ge 0$, $h^{\downarrow}(y) = l f^{\downarrow}\left(\frac{y}{m}\right)$.*

Proof: Let $B(f, y) \stackrel{def}{=} \{x \ge 0, h(x) \ge y\}$ so that $f^{\downarrow}(y) = \inf B(y, f)$. Observe that $x \in B(h, y) \Leftrightarrow \frac{x}{l} \in B\left(f, \frac{y}{m}\right)$. □

Lemma 3.10. *Let $a \in \mathscr{F}$ and $l > 0$. Define $b \in \mathscr{F}$ by $b(x) = l f\left(\frac{x}{l}\right)$. Then, for all $x \ge 0$, $(\lambda_1 \otimes b)(x) = l(\lambda_1 \otimes a)\left(\frac{x}{l}\right)$.*

Do the change of variable $u = l v$ in the expansion $(\lambda_1 \otimes b)(x) = \inf_{0 \le u \le x}(u + b(x-u))$ and obtain $(\lambda_1 \otimes b)(x) = \inf_{0 \le v \le \frac{x}{l}}\left(l v + a\left(\frac{x}{l} - v\right)\right) = l\,(\lambda_1 \otimes a)\left(\frac{x}{l}\right)$. □

We can now compute the lower-pseudo inverse of ψ_i. First, define the sequence g by $g_k = \frac{1}{l_i^{\min}} \psi_i\left(k l_i^{\min}\right)$. As in Lemma 3.8, g can be extended to a piecewise constant function whose lower-pseudo inverse, $g^{\downarrow}$, can be directly computed:

$$g^{\downarrow}(x) = \frac{1}{l_i^{\min}} \sum_{k=0}^{w_i - 1} \nu_{L_{\text{tot}}, l_i^{\min}}\left(l_i^{\min}\left[x - g_k\right]^+\right) \tag{3.28}$$

Second, observe that for all $x \ge 0$, $\psi_i(x) = \psi_i(\lfloor \frac{x}{l_i^{\min}} \rfloor l_i^{\min}) + x \bmod l_i^{\min}$. Define f and h from g as in Lemmas 3.8 and 3.9 with $l = m = l_i^{\min}$, so that $h = \psi_i$. Apply Lemmas 3.8 and 3.9 and obtain $\psi_i^{\downarrow}(x) = l_i^{\min}\left(\lambda_1 \otimes g^{\downarrow}\right)(\frac{x}{l_i^{\min}})$. Now apply Lemma 3.10 with $a = g^{\downarrow}$, $l = l_i^{\min}$, and $b = U_i$ to obtain

$$\psi_i^{\downarrow} = \lambda_1 \otimes U_i \tag{3.29}$$

Proof of Theorem 3.1: Lemma 3.6 gives, in Eq. (3.24), an upper bound on the total amount of service as a function of the service received by the class of interest. We invert Eq. (3.24) by the lower-pseudo inverse technique in Eq. (2.13) and obtain $D_i(t) - D_i(s) \ge \psi_i^{\downarrow}(\beta(t-s))$. The lower-pseudo inverse of ψ_i is given by Eq. (3.29), thus

$$D_i(t) - D_i(s) \ge (\lambda_1 \otimes U_i)\left(\beta\,(t-s)\right) = \beta_i\,(t-s) \tag{3.30}$$

Lastly, we need to prove that β_i is super-additive. This follows from the tightness result in Theorem 3.5 (the proof of which is independent of the rest of this proof). Indeed, the super-additive closure $\bar{\beta}_i$ of β_i is also a strict service curve, and $\bar{\beta}_i(t) \geq \beta_i(t)$ for all t [40, Prop. 5.6]). By Theorem 3.5, we also have $\bar{\beta}_i(t) \leq \beta_i(t)$ for all t, hence $\bar{\beta}_i = \beta_i$. □

3.6.2 Proof of Theorem 3.2

Proof: The WRR strict service curve [40, Sec. 8.2.4] is defined by $\beta'_i(t) = \gamma'_i(\beta(t))$ with

$$\gamma'_i = (\lambda_1 \otimes \nu_{L_{\text{tot}},q_i})\left([t-Q_i]^+\right) \tag{3.31}$$

$$\psi'_i(x) \stackrel{def}{=} x + \sum_{j,j\neq i} \phi'_{i,j}\left(\left\lfloor \frac{x}{l_i^{\min}} \right\rfloor\right) l_j^{\max} \tag{3.32}$$

$$\phi'_{i,j}(x) \stackrel{def}{=} \left(1 + \left\lfloor \frac{x}{w_i} \right\rfloor\right) w_j \tag{3.33}$$

where γ'_i is the lower-pseudo inverse of ψ'_i. We know that for IWRR, γ_i is also the lower-pseudo inverse of ψ_i (defined in Eq. (3.4)). We first show that $\psi_i \leq \psi'_i$.

It is sufficient to prove that for all $j \neq i$ and for all $k \in \mathbb{N}$, $\phi_{i,j}(k) \leq \phi'_{i,j}(k)$. From the definition of $\phi_{i,j}$ and as $\min(x \bmod w_i + 1, w_j) \leq \min(w_i, w_j)$,

$$\phi_{i,j}(x) \leq \left\lfloor \frac{x}{w_i} \right\rfloor w_j + \left[w_j - w_i\right]^+ + \min(w_i, w_j) \tag{3.34}$$

Observe that $\left[w_j - w_i\right]^+ + \min(w_i, w_j) = w_j$. Hence, the right-hand side is $\phi'_{i,j}(x)$. This shows that

$$\psi_i \leq \psi'_i \tag{3.35}$$

In [140, Sec. 10.1], it is shown that

$$\forall f, g \in \mathscr{F}, f \geq g \Rightarrow f^{\downarrow} \leq g^{\downarrow} \tag{3.36}$$

Apply Eq. (3.36) to Eq. (3.35) to conclude the proof. □

3.6.3 Proof of Theorem 3.3

Lemma 3.11. *Consider some integers $w \geq 1$ and $0 \leq k \leq w-1$, a finite sequence $g_0, g_1, \ldots, g_{w-1}$, and a number $a \in \mathbb{R}$ that satisfy*

1. $\forall \ell \in \mathbb{N}$ *if* $0 \leq \ell \leq w-2$ *then* $g_{\ell+1} - g_\ell \geq 1$
2. $\forall \ell \in \mathbb{N}$ *if* $0 \leq \ell \leq w-3$ *then* $g_{\ell+2} - g_{\ell+1} \leq g_{\ell+1} - g_\ell$
3. *if* $k \leq w-2$ *then* $a \geq g_{k+1} - g_k$ *else* $a \geq 1$
4. *if* $k \geq 1$ *then* $a \leq g_k - g_{k-1}$

Define $f\colon [0,w) \to \mathbb{R}$ by $f(x) = g_{\lfloor x \rfloor} + x \bmod 1$ and $h\colon [0,w) \to \mathbb{R}$ by $h(x) = a(x-k) + g_k$. Then $h \geq f$.

Proof: First, we show that

$$\forall \ell \in \{0, \dots, w-1\}, g_k - g_\ell \geq a(k-\ell) \tag{3.37}$$

Case 1: $\ell < k$. Then $g_k - g_\ell = \sum_{k'=\ell}^{k-1} (g_{k'+1} - g_{k'})$. By 2) every term in the sum is $\geq g_k - g_{k-1}$, by 4) is also $\geq a$ and there are $(k-\ell)$ terms, which shows Eq. (3.37).

Case 2: $\ell = k$. Then Eq. (3.37) is obvious.

Case 3: $\ell > k$. Then $g_\ell - g_k = \sum_{k'=k}^{\ell-1} (g_{k'+1} - g_{k'})$. By 2) every term in the sum is $\leq g_{k+1} - g_k$; note that we must have $k \leq w-2$ thus by 3), every term in the sum is also $\leq a$; also, there are $\ell - k$ terms. Thus $g_\ell - g_k \leq a(\ell - k)$, which shows Eq. (3.37) in this case.

We now proceed with the proof of the lemma. Consider some arbitrary $x \in [0, w)$ and let $\ell = \lfloor x \rfloor$. Then

$$f(x) = x - \ell + g_\ell \tag{3.38}$$

$$h(x) = a(x-\ell) + a(\ell - k) + g_k \tag{3.39}$$

$$h(x) - f(x) = \underbrace{(a-1)(x-\ell)}_{A} + \underbrace{g_k - g_\ell - a(k-\ell)}_{B} \tag{3.40}$$

Observe that we must have $a \geq 1$: if $k = w-1$ this follows from 3), and if $k \leq w-2$ it follows from 3) and 1); thus $A \geq 0$. Also $B \geq 0$ by Eq. (3.37). □

Lemma 3.12. *Let $T > 0$ and P a bounded, wide-sense increasing function $[0, T) \to \mathbb{R}$. Extend P to a function $\bar{P} \in \mathscr{F}$ by $\forall x \geq 0, \bar{P}(x) = \left\lfloor \frac{x}{T} \right\rfloor P(T^-) + P(x \bmod T)$ where $P(T^-) \stackrel{def}{=} \sup_{0 \leq t < T} P(t)$.*

Also, consider an affine function L, defined by $L(x) = ax + b$ for some $a \geq \frac{P(T^-)}{T}$ and some $b \in \mathbb{R}$.

If $L(x) \geq P(x)$ for all x in $[0, T)$ then $L \geq \bar{P}$.

Proof: Observe that, for $x \geq 0$, $L(x) = a \left\lfloor \frac{x}{T} \right\rfloor T + L(x \bmod T)$. Now $L(x \bmod T) \geq P(x \bmod T)$ by hypothesis. Thus

$$L(x) \geq a \left\lfloor \frac{x}{T} \right\rfloor T + P(x \bmod T) \tag{3.41}$$

$$\geq \frac{P(T^-)}{T} \left\lfloor \frac{x}{T} \right\rfloor T + P(x \bmod T) = \bar{P}(x) \tag{3.42}$$

□

Lemma 3.13. *Let $f \in \mathscr{F}$ and a rate-latency function $\beta_{r,T}$ such that $r > 0$, $T > 0$, and $\beta_{r,T} \leq f$. Assume that $\beta_{r,T}(x_1) = f(x_1)$ for $x_1 > T$.*

Then there is no other rate-latency function $\beta_{r',T'}$ (i.e., with $(r', T') \neq (r, T)$) such that $\beta_{r,T} \leq \beta_{r',T'} \leq f$.

Proof: Assume that $\beta_{r,T} \leq \beta_{r',T'} \leq f$. The proof consists in showing that $(r, T) = (r', T')$.

First, we know that $\beta_{r,T}(x_1) = f(x_1)$ and $x_1 > T$; thus $r(x_1 - T) = f(x_1)$ and

$$T = x_1 - \frac{f(x_1)}{r} \tag{3.43}$$

Second, observe that we must have $T' \leq T$, since otherwise $\beta_{r,T}(T') > 0 = \beta_{r',T'}(T')$.

Third, observe that $f(x_1) = \beta_{r,T}(x_1) \leq \beta_{r',T'}(x_1) \leq f(x_1)$ thus $\beta_{r',T'}(x_1) = f(x_1)$ and

$$T' = x_1 - \frac{f(x_1)}{r'} \tag{3.44}$$

Combining the last three paragraphs, it follows $x_1 - \frac{f(x_1)}{r'} \leq x_1 - \frac{f(x_1)}{r}$, i.e., $r' \leq r$. Also, we must have $r' \geq r$, since otherwise $\forall x > x_0,\ \beta_{r,T}(x) > \beta_{r',T'}(x)$ with $x_0 = \frac{rT - r'T'}{r - r'}$. Thus, $r' = r$, and it follows from Eq. (3.43) and Eq. (3.44) that $T' = T$. □

Now we proceed with the proof of Theorem 3.3.

1) We first show that $r_k \leq r_{k+1}$ for $k = 0, \ldots, w_i - 2$. Define sequence g by $g_k = \frac{1}{l_i^{\min}} \psi_i \left(k l_i^{\min}\right)$ for $k = 0, \ldots, w_i - 1$. By definition, we have $g_{k+1} - g_k =$

$$1 + \frac{1}{l_i^{\min}} \sum_{j, j \neq i} \left(\min(k+2, w_j) - \min(k+1, w_j)\right) l_j^{\max} \tag{3.45}$$

Observe that $\left(\min(k+2, w_j) - \min(k+1, w_j)\right)$ is equal to 1 if $k+1 < w_j$, and equal to 0 otherwise. Thus, $g_{k+2} - g_{k+1} \leq g_{k+1} - g_k$ for $0 \leq k < w_i - 2$, which shows that $r_k \leq r_{k+1}$ for $k = 0, \ldots, w_i - 3$. Also, observe that $g_{k+1} - g_k \geq 1$, i.e., $r_k \leq 1$, for $0 \leq k \leq w_i - 2$. Hence, $r_{w_i - 2} \leq r_{w_i - 1}$.

2) Let $r \in [r_0^*, r_{k^*}^*]$ and let $T(r)$ be the value of T defined in the Theorem, namely, $T(r) \stackrel{def}{=} \psi_i(k l_i^{\min}) - \frac{k l_i^{\min}}{r}$, where k is defined by $r_{k-1}^* \leq r < r_k^*$ if $r \in [r_0^*, r_{k^*}^*)$ and $k = k^*$ if $r = r_{k^*}^*$. We now show that $\beta_{r,T(r)} \leq \gamma_i$.

We consider two cases: $r_0^* \leq r < r_{k^*}^*$ or $r = r_{k^*}^*$. For the former case, for any r, apply Lemma 3.11 with $w = w_i$, g as defined in 1), k as defined in the paragraph above, and $a = \frac{1}{r}$. As by construction $\frac{1}{r_k} < a \leq \frac{1}{r_{k-1}}$ and $\frac{1}{r_{k-1}} = g_k - g_{k-1}$, 3) and 4) are satisfied. For the latter case, apply again Lemma 3.11 with the same g and $w = w_i$ but now with $k = k^*$ and $a = \frac{1}{r} = \frac{1}{r_{k^*}^*}$. By construction, we have $\frac{1}{r_{k^*}^*} \geq \frac{1}{r_{k^*}} = g_{k^*+1} - g_{k^*}$ and $\frac{1}{r_{k^*}^*} \leq \frac{1}{r_{k^*-1}} = g_{k^*} - g_{k^*-1}$. Thus, conditions 3) and 4) of Lemma 3.11 are satisfied. Let f be the corresponding function f in Lemma 3.11, i.e., $f(x) = g_{\lfloor x \rfloor} + x \bmod 1$ for $0 \leq x < w_i$. Note that for both cases, f is the same. Also, let f_r be the corresponding function h in Lemma 3.11, i.e., $f_r(x) = \frac{1}{r}(x - k) + g_k$ for $0 \leq x < w_i$. By Lemma 3.11, $f_r \geq f$.

Observe that $f(w_i^-) = \frac{1}{l_i^{\min}} \left(\psi_i((w_i - 1) l_i^{\min}) + 1\right) = \frac{1}{l_i^{\min}} \left(w_i l_i^{\min} + \sum_{j, j \neq i} w_j l_j^{\max}\right) = \frac{L_{\text{tot}}}{l_i^{\min}} = \frac{w_i}{r^*}$. Then, as $f_r(x) \geq f(x)$ for $0 \leq x < w_i$ and $\frac{1}{r} \geq \frac{1}{r^*} = \frac{f(w_i^-)}{w_i}$, we can apply Lemma 3.12 with $P = f$ and $L = f_r$. It gives us $\bar{f}$ defined by $\bar{f}(x) = \lfloor \frac{x}{w_i} \rfloor \frac{L_{\text{tot}}}{l_i^{\min}} + f(x \bmod w_i)$ such that $f_r \geq \bar{f}$.

Then, by using Eq. (3.36), $f_r^{\downarrow} \leq \bar{f}^{\downarrow}$. Also, as $\bar{f}^{\downarrow} \geq 0$, we have $\left[f_r^{\downarrow}\right]^+ \leq \bar{f}^{\downarrow}$. Note that for an increasing, linear function L, defined by $\forall x \geq 0, L(x) = ax + b$ with some $a > 0$ and $b > 0$, we have $\left[L^{\downarrow}\right]^+ = \beta_{\frac{1}{a}, b}$; and observe that $f_r(x) = \frac{x}{r} + g_k - \frac{k}{r} = \frac{x}{r} + \frac{T(r)}{l_i^{\min}}$. Hence, $\left[f_r^{\downarrow}\right]^+ = \beta_{r, \frac{T(r)}{l_i^{\min}}}$.

Until now, we have shown that $\beta_{r, \frac{T(r)}{l_i^{\min}}} \leq \bar{f}^{\downarrow}$. Lastly, we show that $l_i^{\min} \bar{f}^{\downarrow}(\frac{x}{l_i^{\min}}) = \gamma_i(x)$ and $l_i^{\min} \beta_{r, \frac{T(r)}{l_i^{\min}}}(\frac{x}{l_i^{\min}}) = \beta_{r,T(r)}(x)$. Observe that $l_i^{\min} \bar{f}(\frac{x}{l_i^{\min}}) = \lfloor \frac{x}{w_i l_i^{\min}} \rfloor L_{\text{tot}} + \psi_i((\frac{x}{l_i^{\min}} \bmod w_i) l_i^{\min})$. Also, $\psi_i(x) = \lfloor \frac{x}{w_i l_i^{\min}} \rfloor L_{\text{tot}} + \psi_i(x \bmod w_i l_i^{\min})$. Hence, we have $\psi_i(x) = l_i^{\min} \bar{f}(\frac{x}{l_i^{\min}})$. By using Lemma 3.9 with $l = m = l_i^{\min}$,

$l_i^{\min} \bar{f}^{\downarrow}(\frac{x}{l_i^{\min}}) = \psi_i^{\downarrow}(x) = \gamma_i(x)$. Also, observe that $l_i^{\min} \beta_{r, \frac{T(r)}{l_i^{\min}}}(\frac{x}{l_i^{\min}}) = \beta_{r,T(r)}(x)$.

Combine the last paragraphs to conclude that $\beta_{r,T(r)} \leq \gamma_i$ for all r in $[r_0^*, r_{k^*}^*]$.

3) We now show that for any $r \in [r_0^*, r_{k^*}^*]$, $\beta_{r,T(r)}$ is a non-dominated lower-bound of γ_i. Let $r' \geq 0$, $T' \geq 0$ such that $\beta_{r,T(r)} \leq \beta_{r',T'} \leq \gamma_i$. We have to show that $r' = r$ and $T' = T(r)$.

First, if r in $[r_0^*, r_{k^*}^*)$, observe that $\beta_{r,T(r)}(x) = \gamma_i(x)$ for $x = \psi_i(kl_i^{\min}) > \psi_i(kl_i^{\min}) - \frac{kl_i^{\min}}{r} = T(r)$. Then, apply Lemma 3.13 with $\beta_{r,T} = \beta_{r,T(r)}$ and $f = \gamma_i$ to conclude that $r' = r$ and $T' = T(r)$.

Second, if $r = r_{k^*}^*$, observe that $\beta_{r,T(r)}(x) = \gamma_i(x)$ for $x = \psi_i(k^* l_i^{\min}) + L_{\text{tot}} > T(r)$. Again, apply Lemma 3.13 with $\beta_{r,T} = \beta_{r,T(r)}$ and $f = \gamma_i$ to conclude that $r' = r$ and $T' = T(r)$.

4) We now show that there is no other non-dominated rate-latency function, $\beta_{r',T'}$, that is upper bounded by γ_i.

First, we must have $T' \geq T(r_0^*)$. This is because $\gamma_i(x) = 0$ for $x \leq \psi_i(0) = T(r_0^*)$.

Second, we must have $r' \geq r_0^*$. Otherwise, we have $r' < r_0^*$ and we previously showed $T' \geq T(r_0^*)$. Thus, $\beta_{r',T'} \leq \beta_{r_0^*,T(r_0^*)} \leq \gamma_i$, which is in contradiction with $\beta_{r',T'}$ being non-dominated.

Third, we must have $r' \leq r_{k^*}^*$. We proceed to prove this by contradiction. If $T' \geq T(r_{k^*}^*)$ and $r' > r_{k^*}^*$, observe that $\beta_{r',T'}(x_0) = \beta_{r_{k^*}^*,T(r_{k^*}^*)}(x_0)$ with $x_0 = \frac{r'T' + r_{k^*}^* T(r_{k^*}^*)}{r' - r_{k^*}^*}$ and $\forall x, x > x_0 \Rightarrow \beta_{r',T'}(x) > \beta_{r_{k^*}^*,T(r_{k^*}^*)}(x)$; for any arbitrary, non-negative integer k, let x_k be defined by $x_k = \psi_i(k^* l_i^{\min}) + kL_{\text{tot}}$. Then observe that $\beta_{r_{k^*}^*,T(r_{k^*}^*)}(x_k) = \gamma_i(x_k)$. Choose some k large enough such that $x_k > x_0$; then, $\beta_{r',T'}(x_k) > \beta_{r_{k^*}^*,T(r_{k^*}^*)}(x_k) = \gamma_i(x_k)$, which is in contradiction with $\beta_{r',T'} \leq \gamma_i$. Also, if $T' < T(r_{k^*}^*)$ and $r' > r_{k^*}^*$, we have $\forall x, x > T' \Rightarrow \beta_{r',T'}(x) > \beta_{r_{k^*}^*,T(r_{k^*}^*)}(x)$. Choose some k large enough such that $x_k > T'$; then, $\beta_{r',T'}(x_k) > \beta_{r_{k^*}^*,T(r_{k^*}^*)}(x_k) = \gamma_i(x_k)$, which is in contradiction with $\beta_{r',T'} \leq \gamma_i$. Therefore, $r' > r_{k^*}^*$ is in contradiction with $\beta_{r',T'} \leq \gamma_i$.

Therefore, we must have r' in $[r_0^*, r_{k^*}^*]$. We now show that $T' = T(r')$. Because otherwise, if $T' < T(r')$, we have $\beta_{r',T(r')} \leq \beta_{r',T'} \leq \gamma_i$, which is in contradiction with $\beta_{r',T(r')}$ being a non-dominated rate latency function. Also, if $T' > T(r')$, we have $\beta_{r',T'} \leq \beta_{r',T(r')} \leq \gamma_i$, which is in contradiction with $\beta_{r',T'}$ being non-dominated. □

3.6.4 Proof of Theorem 3.4

Let us call the supremum of all non-dominated rate-latency functions B. We want to show that $B = \max\left(\beta_{r_0^*,T_0^*}, \ldots, \beta_{r_{k^*}^*,T_{k^*}^*}\right)$. The proof consists on three steps.

1) $B(x) = 0$ for all x in $[0, l_i^{\min} g_0]$.

2) $B(x) = \beta_{r_{k-1}^*,T_{k-1}^*}(x)$ for all x in $[l_i^{\min} g_{k-1}, l_i^{\min} g_k]$ and $k = 1 \ldots k^*$.

3) $B(x) = \beta_{r_{k^*}^*,T_{k^*}^*}(x)$ for all $x \geq l_i^{\min} g_{k^*}$.

To prove 1), as every non-dominated rate-latency function is equal to zero before $l_i^{\min} g_0$, we have $B(x) = 0$ for all x in $[0, l_i^{\min} g_0]$.

To prove 2), we consider two cases for any other non-dominated rate-latency $\beta_{r',T(r')}$: First, $r_0^* \leq r' <$

r^*_{k-1}. Second, $r^*_{k-1} < r' \leq r^*_{k^*}$.

For the former case, we show that

$$\beta_{r',T(r')}\left(l_i^{\min} g_{k-1}\right) \leq \beta_{r^*_{k-1},T^*_{k-1}}\left(l_i^{\min} g_{k-1}\right) \tag{3.46}$$

Then, as $r' < r^*_{k-1}$, it follows $\beta_{r',T(r')}(x) \leq \beta_{r^*_{k-1},T^*_{k-1}}(x)$ for all x in $[l_i^{\min} g_{k-1}, l_i^{\min} g_k]$.

Let k' defined by $r' \in [r^*_{k'-1}, r^*_{k'})$. Then, by definition

$$\begin{aligned}
\beta_{r',T(r')}\left(l_i^{\min} g_{k-1}\right) &= r'\left(l_i^{\min} g_{k-1} - l_i^{\min} g_{k'}\right) + k' l_i^{\min} && (3.47)\\
&= r'\left(\sum_{e=k'}^{k-2} g_{e+1} - g_e\right) l_i^{\min} + k' l_i^{\min} && (3.48)\\
&\leq r^*_{k'}\left(\sum_{e=k'}^{k-2} g_{e+1} - g_e\right) l_i^{\min} + k' l_i^{\min} && (3.49)\\
&= \frac{\sum_{e=k'}^{k-2} g_{e+1} - g_e}{g_{k'+1} - g_{k'}} l_i^{\min} + k' l_i^{\min} && (3.50)
\end{aligned}$$

Then, as $g_{e+1} - g_e$ is decreasing, we have $\sum_{e=k'}^{k-2} g_{e+1} - g_e \leq \left(k-1-k'\right)\left(g_{k'+1} - g_{k'}\right)$. Combine it with Eq. (3.50) to conclude that $\beta_{r',T(r')}\left(l_i^{\min} g_{k-1}\right) \leq (k-1)\, l_i^{\min}$; lastly, observe that $\beta_{r^*_{k-1},T^*_{k-1}}\left(l_i^{\min} g_{k-1}\right) = (k-1)\, l_i^{\min}$. Therefore, Eq. (3.46) is proven.

For the latter case, we show that

$$\beta_{r',T(r')}\left(l_i^{\min} g_k\right) \leq \beta_{r^*_{k-1},T^*_{k-1}}\left(l_i^{\min} g_k\right) \tag{3.51}$$

Then, as $r' > r^*_{k-1}$, it follows $\beta_{r',T(r')}(x) \leq \beta_{r^*_{k-1},T^*_{k-1}}(x)$ for all x in $[l_i^{\min} g_{k-1}, l_i^{\min} g_k]$.

Let k' defined by $r' \in [r^*_{k'-1}, r^*_{k'})$. Then, by definition

$$\begin{aligned}
\beta_{r',T(r')}\left(l_i^{\min} g_k\right) &= r'\left(l_i^{\min} g_k - l_i^{\min} g_{k'}\right) + k' l_i^{\min} && (3.52)\\
&= r'\left(-\sum_{e=k}^{k'-1} g_{e+1} - g_e\right) l_i^{\min} + k' l_i^{\min} && (3.53)\\
&\leq r^*_{k'}\left(-\sum_{e=k}^{k'-1} g_{e+1} - g_e\right) l_i^{\min} + k' l_i^{\min} && (3.54)\\
&\leq \frac{-\sum_{e=k}^{k'-1} g_{e+1} - g_e}{g_{k'+1} - g_{k'}} l_i^{\min} + k' l_i^{\min} && (3.55)
\end{aligned}$$

Then, as $g_{e+1} - g_e$ is decreasing, we have $-\sum_{e=k}^{k'-1} g_{e+1} - g_e \leq \left(k-k'\right)\left(g_{k'+1} - g_{k'}\right)$. Combine it with Eq. (3.55) to conclude that $\beta_{r',T(r')}\left(l_i^{\min} g_k\right) \leq k l_i^{\min}$; lastly, observe that $\beta_{r^*_{k-1},T^*_{k-1}}\left(l_i^{\min} g_k\right) = k l_i^{\min}$. Therefore, Eq. (3.51) is proven.

Combining these two cases, 2) is proven.

To prove 3), for any other non-dominated rate-latency $\beta_{r',T(r')}$, we show that

$$\beta_{r',T(r')}\left(l_i^{\min} g_{k^*}\right) \leq \beta_{r^*_{k^*},T^*_{k^*}}\left(l_i^{\min} g_{k^*}\right) \tag{3.56}$$

Then, as $r' < r^*_{k^*}$, it follows $\beta_{r',T(r')}(x) \leq \beta_{r^*_{k^*},T^*_{k^*}}(x)$ for all $x \geq l_i^{\min} g_{k^*}$.

Let k' defined by $r' \in [r^*_{k'-1}, r^*_{k'})$. Then, by definition

$$\beta_{r',T(r')}\left(l_i^{\min} g_{k^*}\right) = r'\left(l_i^{\min} g_{k^*} - l_i^{\min} g_{k'}\right) + k' l_i^{\min} \tag{3.57}$$

$$= r'\left(\sum_{e=k'}^{k^*-1} g_{e+1} - g_e\right) l_i^{\min} + k' l_i^{\min} \tag{3.58}$$

$$\leq r^*_{k'}\left(\sum_{e=k'}^{k^*-1} g_{e+1} - g_e\right) l_i^{\min} + k' l_i^{\min} \tag{3.59}$$

$$\leq \frac{\sum_{e=k'}^{k^*-1} g_{e+1} - g_e}{g_{k'+1} - g_{k'}} l_i^{\min} + k' l_i^{\min} \tag{3.60}$$

Then, as $g_{e+1} - g_e$ is decreasing, we have $\sum_{e=k'}^{k^*-1} g_{e+1} - g_e \leq \left(k^* - k'\right)\left(g_{k'+1} - g_{k'}\right)$. Combine it with Eq. (3.60) to conclude that $\beta_{r',T(r')}\left(l_i^{\min} g_{k^*}\right) \leq k^* l_i^{\min}$; lastly, observe that $\beta_{r^*_{k^*},T^*_{k^*}}\left(l_i^{\min} g_{k^*}\right) = k^* l_i^{\min}$. Therefore, Eq. (3.56) is proven.

Until now, we have shown **1)**, **2)**, and **3)**. Let $A = \max\left(\beta_{r^*_0,T^*_0}, \ldots, \beta_{r^*_{k^*},T^*_{k^*}}\right)$. Observe that first, by **1)**, it follows $A = 0$ for all x in $[0, l_i^{\min} g_0]$; second, by **2**, it follows $A(x) = \beta_{r^*_{k-1},T^*_{k-1}}(x)$ for all x in $[l_i^{\min} g_{k-1}, l_i^{\min} g_k]$ and $k = 1 \ldots k^*$; lastly, by **3)**, $A(x) = \beta_{r^*_{k^*},T^*_{k^*}}(x)$ for all $x \geq l_i^{\min} g_{k^*}$. Therefore, $A = B$, i.e., $B = \max\left(\beta_{r^*_0,T^*_0}, \ldots, \beta_{r^*_{k^*},T^*_{k^*}}\right)$.

We now want to show that B is the largest convex function upper bounded by γ_i, i.e., if f is a convex function and is upper bounded by γ_i, then $f \leq B$.

Pick an arbitrary $x \geq 0$. Let G_x be a subgradient of f at x. Note that a subgradient exists because f is convex [153, Sec. 5.4]. By definition of subgradient [153, Sec. 5.4], for L, defined by $\forall x' \geq 0, L(x') = G_x(x' - x) + f(x)$, we have $L \leq f$; then, as $f \leq \gamma_i$, we have $L \leq \gamma_i$. We now consider two cases for G_x and proceed with the proof to show that $f(x) \leq B(x)$ in both cases.

Case 1: $G_x \leq 0$

As $L(0) \leq \gamma_i(0) = 0$, we have $L \leq 0$; also, observe that $B \geq 0$. Hence, $L \leq B$. It follows $L(x) = f(x) \leq B(x)$.

Case 1: $G_x > 0$

Define $\beta_{r,T}$ with $r = G_x$ and $T = x - \frac{f(x)}{G_x}$. Observe that $r \geq 0$ and as $L(0) \leq \gamma_i(0) = 0$, it follows $T \geq 0$. We now proceed to show that $\beta_{r,T} \leq \gamma_i$. As $L \leq \gamma_i$ and $\gamma_i \geq 0$, we have $[L]^+ \leq \gamma_i$; also, observe that $[L]^+ = \beta_{r,T}$. Therefore, $\beta_{r,T} \leq \gamma_i$.

Then, as $\beta_{r,T}$ is a rate-latency function upper bounded by γ_i, it is dominated by one of the non-dominated rate-latencies or is equal to one of them. It follows $\beta_{r,T} \leq B$; also, observe that $\beta_{r,T}(x) = f(x)$. Thus, $f(x) \leq B(x)$.

Lastly, the above result applies to any $x \geq 0$, thus $\forall x \geq 0, f(x) \leq B(x)$, i.e., $f \leq B$. □

3.6.5 Proof of Theorem 3.5

We use the following lemma about the lower pseudo-inverse technique.

Lemma 3.14. *For a right-continuous function f in $\mathscr{F}$ and x, y in $\mathbb{R}^+$, $f^{\downarrow}(y) = x$ if and only if $f(x) \geq y$*

and there exists some $\varepsilon > 0$ *such that* $\forall x' \in (x-\varepsilon, x), f(x') < y$.

Proof:

$\Rightarrow$:

Let $S = \{x', f(x') \geq y\}$ so that $x = \inf S$ (see Eq. (2.12)). From the definition of an inf, there exists a sequence x_n such that $x_n \in S$ for all n, $x_n \geq x$, and $\lim_{n\to\infty} x_n = x$. Since f is right-continuous, $\lim_{n\to\infty} f(x_n) = f(x)$, which shows that $f(x) \geq y$. Also, again by definition of an inf, any $x' < x$ does not belong to S, i.e. $\forall x' < x, f(x') < y$.

$\Leftarrow$:

By the first part of the hypothesis, $x \in S$ therefore $x \geq \inf S = f^{\downarrow}(y)$. Let also $S' = \{x', f(x') < y\}$ so that $f^{\downarrow}(y) = \sup S'$ (see Eq. (2.12)). By the second part of the hypothesis, S' contains the interval $(x-\varepsilon, x)$ hence $\sup S' \geq x$, which shows that $f^{\downarrow}(y) \geq x$. Combining the two shows that $f^{\downarrow}(y) = x$. □

Proof of Theorem 3.5

We prove that, for any value of the system parameters, for any $\tau > 0$, and for any class i, there exists one trajectory of a system such that

$$\begin{aligned} &\exists s \geq 0, (s, s+\tau] \text{ is backlogged for class } i \\ &\text{and } D_i(s+\tau) - D_i(s) = \beta_i(\tau) \end{aligned} \tag{3.61}$$

Step 1: Constructing the Trajectory

1) Classes are labeled in order of weights, i.e., $w_j \leq w_{j+1}$.

2) At time 0, the input of every queue $j \neq i$ is a burst of size $\left\lceil \frac{\beta(\tau)}{l_j^{\max}} \right\rceil l_j^{\max} + w_j l_j^{\max}$.

3) Every class, $j \neq i$, is packetized according to its maximum packet size, $l_j^{\max}$.

4) The output of the system is at rate K (the Lipschitz constant of β) from time 0 to times s, which is defined as the time at which queue i is visited at cycle w_i in the first round, namely

$$s = \frac{1}{K} \sum_{j, j\neq i} \min\left(w_i - 1, w_j\right) l_j^{\max} \tag{3.62}$$

It follows that

$$\forall t \in [0, s], D(t) = Kt \tag{3.63}$$

5) The input of queue i starts just after time s, with a burst of size $\left\lceil \frac{\beta(\tau)}{l_i^{\min}} \right\rceil l_i^{\min}$.

6) Class i is packetized according to its minimum packet size, $l_i^{\min}$.

7) After time s, the output of the system is equal to the guaranteed service; by 2) and 5), the busy period lasts for at least τ, i.e.,

$$\forall t \in [s, s+\tau], D(t) = D(s) + \beta(t-s) \tag{3.64}$$

In particular,

$$D(s+\tau) - D(s) = \beta(\tau) \tag{3.65}$$

If we apply $\psi_i^{\downarrow}$ to both sides of Eq. (3.65), the right-hand side is equal to $\beta_i(\tau)$. Thereby, we should prove

$$\psi_i^{\downarrow}(D(s+\tau) - D(s)) = D_i(s+\tau) - D_i(s) \tag{3.66}$$

Let $y = D(s+\tau) - D(s)$ and $x = D_i(s+\tau) - D_i(s)$. Our goal is now to prove that

$$\psi_i^{\downarrow}(y) = x \tag{3.67}$$

From 5), we know that the first packet of class i is served at the first cycle of a round ($C = 1$ in Algorithm 3.1). Thus, applying Lemma 3.2 and (P5) in Lemma 3.3, the number of services to each class j is equal to $\phi_{i,j}(p)$. From 2), class j sends packets with the maximum length. Thus

$$\sum_{j,j\neq i} D_j(s+\tau_{\sigma(p)}) - D_j(s) = \sum_{j,j\neq i} \phi_{i,j}(p) l_j^{\max} \tag{3.68}$$

Now there are two cases for $s+\tau$ (3.6.1.1).

Case 1: $s+\tau < \tau_{\sigma(p)}$ In this case the scheduler is not serving class i in $[\tau_{\sigma(p)}, s+\tau]$ and $x = p l_i^{\min}$. Thus $D_i(s+\tau) = D_i(\tau_{\sigma(p)})$. It follows that

$$\begin{aligned} \psi_i(x) &= x + \underbrace{\sum_{j,j\neq i} \phi_{i,j}(\lfloor \frac{x}{l_i^{\min}} \rfloor) l_j^{\max}}_{\sum_{j,j\neq i} D_j(\tau_{\sigma(p)}) - D_j(s)} \\ y &= x + \sum_{j,j\neq i} D_j(s+\tau) - D_j(s) \end{aligned} \tag{3.69}$$

and thus

$$\psi_i(x) \geq y \tag{3.70}$$

Let $x - l_i^{\min} < x' < x$; class i's output becomes equal to x' during the emission of packet $p-1$ thus

$$\psi_i(x') = x' + \sum_{j,j\neq i} D_j(\tau_{\sigma(p-1)}) - D_j(s) \tag{3.71}$$

Hence

$$\forall x' \in (x - l_i^{\min}, x), \psi_i(x') < y \tag{3.72}$$

Combining Eq. (3.70) and Eq. (3.72) with Lemma 3.14 shows Eq. (3.67).

Case 2: $s+\tau \geq \tau_{\sigma(p)}$ In this case the scheduler is serving class i in $[\tau_{\sigma(p)}, s+\tau]$. For every other class j, we have $D_j(s+\tau) = D_j(\tau_{\sigma(p)})$. Hence,

$$\psi_i(x) = D_i(s+\tau) - D_i(s) + \sum_{j,j\neq i} \phi_{i,j}(p) l_j^{\max} = y \tag{3.73}$$

As with case 1, for any $x' \in ((p-1) l_i^{\min}, x)$, we have $\psi_i(x) < y$, which shows Eq. (3.67).

This shows that Eq. (3.61) holds. It remains to show that the system constraints are satisfied.

Step 2: Verifying the Trajectory

We need to verify that the service offered to the aggregate satisfies the strict service curve constraint. Our trajectory has one busy period, starting at time 0 and ending at some time $T_{\max} \geq \tau$. We need to verify that

$$\forall t_1, t_2 \in [0, T_{\max}] \text{ with } t_1 < t_2, D(t_2) - D(t_1) \geq \beta(t_2 - t_1) \tag{3.74}$$

Case 1: $t_2 < s$

Then $D(t_2) - D(t_1) = K(t_2 - t_1)$. Observe that, by the Lipschitz continuity condition on β, for all $t \geq 0$, $\beta(t) = \beta(t) - \beta(0) = \beta(t) \leq Kt$ thus $K(t_2 - t_1) \geq \beta(t_2 - t_1)$.

Case 2: $t_1 < s \leq t_2$

Then $D(t_2) - D(t_1) = \beta(t_2 - s) + K(s - t_1)$. By the Lipschitz continuity condition:

$$\beta(t_2 - t_1) - \beta(t_2 - s) \leq K(s - t_1) \tag{3.75}$$

thus $D(t_2) - D(t_1) \geq \beta(t_2 - t_1)$.

Case 3: $s \leq t_1 < t_2$

Then $D(t_2) - D(t_1) = \beta(t_2) - \beta(t_1) \geq \beta(t_2 - t_1)$ because β is super-additive. □

3.6.6 Proof of Theorem 3.6

Proof: The proof is very similar to the proof of Theorem 3.5. The necessary changes in the proof are the following:

1) s is the time of the first visit to class i.

2) Instead of functions ψ_i and $\phi_{i,j}$, use functions ψ'_i and $\phi'_{i,j}$, defined in Eq. (3.32) and Eq. (3.33). □

3.6.7 Proof of Theorem 3.7

Proof: The proof contains the following steps:

1) Consider the same trajectory as in the proof of Theorem 3.5, yet with one difference: the input of class i is $A_i(t) = \alpha_i(t - s)$ for $t \geq s$ and zero before s. Observer that as α_i is sub-additive, $\forall t_1, t_2$: $t_2 \geq t_1 \geq s \Rightarrow A_i(t_2) - A_i(t_1) = \alpha_i(t_2) - \alpha_i(t_1) \leq \alpha_i(t_2 - t_1)$.

2) Define $s' = \inf\{u > 0 | \alpha_i(u) \leq \beta_i(u)\}$. This is the first time after zero that the service curve meets the arrival curve. Note that s' can be infinite as well.

3) Then, it is guaranteed that class i is backlogged in $(s, s + s']$. Therefore, using Eq. (3.61), we have $D_i(t) = \beta_i(t - s)$ for $t \geq s$ and zero before s.

4) Combining 1 and 3, the horizontal deviation of A_i and D_i in $(s, s + s']$ is equal to the horizontal deviation of α_i and β_i in $[0, s']$.

4) Using [40, Sec. 5.3.3], the horizontal deviation of α_i and β_i can be restricted to $[0, s']$.

Thereby, we find a valid trajectory (verified in the proof of Theorem 3.5) where the delay bound is achieved. □

3.6.8 Proof of Theorem 3.8

Proof: The same proof of Theorem 3.7 works here as well. However, we use the trajectory defined in the proof of Theorem 3.6. □

3.7 Conclusion

IWRR is a variant of WRR with the same long-term rate and the same complexity. We have provided a residual strict service curve for IWRR and have shown that it is the best possible one under general assumptions. For classes with packets of constant size, we have shown that the delay bounds derived from it are worst-case. We have proved that IWRR worst-case delay is not greater than WRR and shown on experiments that the gain is significant (20 %–60 %) in practice, which speaks in favor of using IWRR as a replacement to WRR. Our result assumes that the aggregate of all IWRR queues receives a strict service curve guarantee, and we find a strict service curve guarantee for every IWRR queue. Therefore, our results apply to hierarchical schedulers.

3.8 Notation

Table 3.1: Notation List, Specific to Chapter 3

i	A class
A	Aggregate, cumulative arrival function of all classes
D	Aggregate, cumulative departue function of all classes
α_i	An arrival curve for class i
β_i	A strict service curve offered to class i
β	A strict service curve offered to aggregate of all classes
A_i	Cumulative arrival function of class i
D_i	Cumulative departure function of class i
n	Number of classes
$l_i^{\max}$	Maximum packet size for flows of class i
$l_i^{\min}$	Minimum packet size for flows of class i
w_i	Weight of class i
L_{tot}	$w_i l_i^{\min} + \sum_{j,j\neq i} w_j l_j^{\max}$
λ_c	Rate function with $\lambda_c(t) = ct$
$\beta_{R,L}$	Rate-latency function with $\beta_{c,L}(t) = \max(0, c(t-L))$
$\mathbb{R}^+$	Set of non-negative real numbers
$\mathscr{F}$	Set of wide-sense increasing functions $f : \mathbb{R}^+ \mapsto \mathbb{R}^+ \cup \{+\infty\}$
$\nu_{p,b}$	Stair function with $\nu_{p,b}(t) = b\left\lceil \frac{t}{p} \right\rceil$
$\gamma_{r,b}$	Token-bucket function with $\gamma_{r,b}(0) = 0$ and $\gamma_{r,b}(t) = rt + b$ for $t > 0$
$[x]^+$	$[x]^+ = \max(0, x)$
$\lfloor x \rfloor$	Flooring function
$f^{\downarrow}$	Lower pseudo inverse $f^{\downarrow} = \inf\{x \mid f(x) \geq y\} = \sup\{x \mid f(x) < y\}$
$\otimes$	Min-plus convolution $(f \otimes g)(t) = \inf_{0\leq s\leq t}\{f(t-s) + g(s)\}$

4 Strict Service Curves for Deficit Round-Robin

In the realm of queues, where fairness prevails,
Deficit Round-Robin, its method unveils.
With packets variable, it stands strong,
Providing fair scheduling all along.

Worst-case delays, Boyer's bounds did find,
Network calculus, a rigorous kind.
A convex strict curve, service defined,
For one class of interest, so refined.

But assumptions lacking, when traffic interferes,
Pessimism arises, fueling our fears.
Soni proposed a correction, a semi-rigor guide,
Yet alas, incorrect, a counter-example beside.

Bouillard stepped in, with rigor embraced,
Convex strict curves, arrival curves placed.
Improvements we seek, our mission prevails,
In two ways we delve, advancing the trails.

A non-convex curve, DRR refined,
No arrival constraint, its greatness defined.
An iterative method, constraints to appease,
Improving all curves, bringing solace and ease.

Today, our results stand tall and profound,
The best-known bounds, DRR's honor renowned.
Pseudo-inverse method, our ally and guide,
In the pursuit of knowledge, we stride.

So let this chapter unfold, a journey untold,
As DRR's secrets, we begin to behold.

Created with ChatGPT, free research preview (version May 24) [141]

As explained in Section 1.2.2, although WRR and IWRR were originally designed in the context of ATM [85] with constant-size packets to share the bandwidth in proportion to allocated weights, they

have been applied to networks with variable-length packets. In such cases, the bandwidth allocated to each queue depends on not only the weights but also on the packet sizes. This is not desirable, as the intention of the weights is to control the allocated bandwidth to each queue; however, they do not entirely control this as the packet sizes interfere. Deficit Round-Robin (DRR) [154] is a later variant that solves this. DRR is often used for scheduling tasks, or packets, in real-time systems or communication networks. With DRR, every queue is associated with a static number, called *quantum*. Queues are visited *round-robin* (one after the other), and at every visit, receive service (measured in bits for communication networks, in seconds for task processing systems) up to the quantum value. Tasks or packets are of variable sizes, and it may happen that, during one visit of the server, there remains at least one task or packet in the queue that cannot be served because the unused part of the quantum is positive but not large enough. In such a case, the unused part of the quantum (called the *residual deficit*) is carried over to the next round. DRR shares resources flexibly (the amount of service reserved for one queue is proportional to its quantum) and efficiently (when a queue is idle, the server capacity is available to other queues). It is widely used as it has low complexity and very efficient implementations exist [80].

DRR can be applied to time-sensitive networks, i.e., to communication networks where it is required to obtain bounds on worst-case delay (not on average). Here, flows are grouped in some classes, and every class corresponds to one DRR queue at every node. Furthermore, the traffic that every flow can send is limited at the source by an arrival curve constraint, i.e., a limit to the number of bits that can be sent over any time interval. Worst-case delay bounds for such a setting were obtained in [86, 87, 88] using various ad-hoc analyses. These results were improved in [89], where the authors obtain a strict service curve for DRR , i.e., a function that lower-bounds the amount of service received by every DRR queue. A strict service curve is a special case of a service curve hence can be used to derive delay (and backlog) bounds (see Section 2.1.1.4). We call this method the *strict service curve of Boyer et al.*

In a DRR system, if a queue does not have enough traffic to use its quantum at every visit of the server, then the leftover capacity is automatically used to improve the service received by other queues. In a time-sensitive network, some or all interfering traffic is deterministic, and in normal operation, is limited at the source. There is interest in obtaining proven bounds for both the degraded operational mode (when some traffic classes misbehave) and for the non-degraded mode (when all time-sensitive traffic satisfies its source constraints). The strict service curve of Boyer et al. does not make any assumptions on the interfering traffic. Hence, the resulting delay bounds are valid, even in degraded operational mode.

For the non-degraded operational mode, i.e., when arrival curve constraints can be assumed for interfering traffic, significantly smaller delay bounds were presented at a RTSS conference [90]. They use the result of [89], which is improved by what we call the *correction term of Soni et al.* Unfortunately, the method is semi-rigorous and cannot be fully validated. Indeed, our first contribution is to show that the correction term of Soni et al. is incorrect; we do so by exhibiting a counter-example that satisfies their assumptions and that has a larger delay (Section 4.3). The main idea of the correction term of Soni et al. is the assumption that only packets of interfering flows arriving within a duration of the delay bound of Boyer et al. will get a chance to delay a given packet of the flow of interest; in our counter-example, we observed that this assumption is incorrect, and all packets of interfering flows arriving within the global backlogged period might delay a packet of the flow of interest. Later, Bouillard, in [91], derived new strict service curves for DRR that account for the arrival curve constraints of the interfering traffic and improve on the strict service curve of Boyer et al., hence on the delay bounds. These results are formally proven. They require that arrival curves are concave (which does

not always hold, e.g., when sources are periodic).

Our next contribution is obtaining a better strict service curve for DRR when we do not take into account arrival curve constraint on interfering traffic, i.e., for degraded operational mode. To do so, we rely on the method of pseudo-inverse, as it enables us to capture all details of DRR ; we used a similar method to obtain a strict service curve for Interleaved Weighted Round-Robin (IWRR) in Chapter 3. We also provide simplified lower bounds that can be used when analytic, closed-form expressions are important. One such lower bound is precisely the strict service curve of Boyer et al. (Fig. 4.2), hence the worst-case delay bounds obtained with our strict service curve are guaranteed to be less than or equal to those of Boyer et al.

Our following contribution is a new iterative method for obtaining better strict service curves for DRR that account for the arrival curve constraints of interfering flows. Our method is rigorous and is based on pseudo-inverses and output arrival curves of interfering flows. We also provide simpler variants. Our method improves on any available strict service curves for DRR, hence, we always improve on Bouillard's strict service curve. Furthermore, our method accepts any type of arrival curves, including non-concave ones (such as the stair function used with periodic flows), and can be applied to any type of strict service curve, including non-convex ones (such as the strict service curve we obtained when there is no arrival curve constraint on interfering traffic).

The delay bounds obtained with our method are fully proven. Furthermore, we compute them for the same case studies as in Bouillard's work [91] (one single server analysis) and as in Soni et al. [90] (including two illustration networks and an industrial-sized one). We find that they are smaller than Bouillard's and the incorrect ones that use the correction term of Soni et al. Hence as of today, it follows that our delay bounds are the best-proven delay bounds for DRR, with or without constraints on interfering traffic.

The rest of this chapter is organized as follows. After giving related works in Section 4.2, we describe the counter-example to Soni et al. in Section 4.3. In Section 4.4, we present our new strict service curves for DRR, with no knowledge of interfering traffic. In Section 4.5, we present our new strict service curves for DRR; they account for the interfering arrival curve constraints. In Section 4.6, we use numerical examples to illustrate the improvement in delay bounds obtained with our new strict service curves. We present proofs of results in Section 4.7. We conclude the chapter in Section 4.8. A summary of notation and symbols used in this chapter are given in Section 4.9.

4.1 System Model

We consider a DRR system in the context of deterministic networking, and we are interested in the worst-case delays for flows, given arrival curve constraints on the flows. A DRR subsystem serves n inputs, has one queue per input, and uses Algorithm 4.1 for serving packets. Each queue i is assigned a quantum Q_i. DRR runs an infinite loop of *rounds*. In one round, if queue i is non-empty, a service for this queue starts and its *deficit* is increased by Q_i. The service ends when either the deficit is smaller than the head-of-the-line packet or the queue becomes empty. In the latter case, the deficit is set back to zero. The `send` instruction is assumed to be the only one with a non-null duration. Its actual duration depends on the packet size but also on the amount of service available to the entire DRR subsystem.

Algorithm 4.1: Deficit Round-Robin

```
   Input: Integer quantum Q_1, Q_2, ..., Q_n
 1 for i ← 1 to n do
       //Deficits are initially zero.
 2     d_i ← 0;
 3 while true do
 4     for i ← 1 to n do
 5         if (not empty(i)) then
               //A service for queue i starts.
 6             d_i ← d_i + Q_i;
 7             while (not empty(i)) and (size (head(i)) ≤ d_i) do
 8                 d_i ← d_i − size (head(i));
 9                 send(head(i));
10                 removeHead(i);
               //A service for queue i ends.
11             if (empty(i)) then
12                 d_i ← 0;
```

In [89] as in much of the literature on DRR, the set of packets that use a given queue is called a *flow*; a flow may, however, be an aggregate of multiple flows, called micro-flows [155] and an aggregate flow is called a *class* in [90]. In order to be consistent with the rest of this thesis, we consider that a DRR input corresponds to one class.

The DRR subsystem is itself placed in a larger system and can compete with other queuing subsystems. A common case is when the DRR subsystem is at the highest priority on a non-preemptive server with line rate c. Due to non-preemption, the service offered to the DRR subsystem might not be instantly available. This can be modeled by means of a rate-latency strict service curve (see Section 2.1 for the definition), with rate c and latency $\frac{c}{L^{\max}}$ where $L^{\max}$ is the maximum packet size of lower priority. If the DRR subsystem is not at the highest priority level, this can be modeled with a more complex strict service curve [40, Section 8.3.2]. This motivates us to assume that the aggregate of all flows in the DRR subsystem receives a strict service curve β, which we call "aggregate strict service curve". If the DRR subsystem has exclusive access to a transmission line of rate c, then $\beta(t) = ct$ for $t \geq 0$. We assume that $\beta(t)$ is finite for every (finite) t. Note that the aggregate strict service curve β, which models the service offered to the aggregate of all flows, should not be confused with the strict service curves such as β_i^m, which captures service offered to class i; strict service curves such as β_i^m are called "residual" strict service curves in [89].

Here, we use the language of communication networks, but the results equally apply to real-time systems: Simply map flow to task, map packet to job, map packet size to job-execution time, and map strict service curve to "delivery curve" [142, 143].

4.2 Related Works

4.2.1 Strict Service Curve of Boyer et al.

The strict service curve of Boyer et al. for DRR is given in [89], and we rewrite it using our notation. For class i, let $d_i^{\max}$ be its maximum residual deficit, defined by $d_i^{\max} = l_i^{\max} - \epsilon$ where $l_i^{\max}$ is an upper bound

on the packet size and ϵ is the smallest unit of information seen by the scheduler (e.g., one bit, one byte, one 32-bit word, ...). Also, let $Q_{\text{tot}} = \sum_{j=1}^{n} Q_j$. Then, for every class i, their strict service curve is the rate-latency service curve $\beta_{R_i,T_i}\left(\beta(t)\right)$ with rate $R_i = \frac{Q_i}{Q_{\text{tot}}}$ and latency $T_i = \sum_{j\neq i} d_j^{\max} + (1 + \frac{d_i^{\max}}{Q_i}) \sum_{j\neq i} Q_j$ (see Section 2.1.1.4 for the definition of a rate-latency function). This rate-latency is illustrated in Fig. 4.2 (the red curve); Note that the latency is equal to the maximum service interruption plus additional terms. As Boyer et al. find a rate-latency with the maximum rate, they were forced to take a latency larger than the maximum service interruption; this explains why it is not optimal.

4.2.2 Correction Term of Soni et al.

When interfering classes are constrained by some arrival curves, Soni et al. give a correction term that improves the obtained delay bounds using the strict service curve of Boyer et al. in [90], which we now rewrite using our notation. Assume that every class i has an arrival curve α_i, and the server is a constant-rate server with a rate equal to c. Let $D_i^{\text{Boyer-et-al}}$ be the network calculus delay bound for class i obtained by combining α_i with the strict service curve of Boyer et al., as explained in Section 4.2.1. The delay bound proposed in [90] is $D_i^{\text{Soni-et-al}} = D_i^{\text{Boyer-et-al}} - C_i^{\text{Soni-et-al}}$ with

$$C_i^{\text{Soni-et-al}} = \sum_{j,j\neq i} \frac{\left[S_j(D_i^{\text{Boyer-et-al}}) - \alpha_j(D_i^{\text{Boyer-et-al}})\right]^+}{c} \tag{4.1}$$

where $S_j(t) \stackrel{def}{=} \left(Q_j + d_j^{\max}\right) 1_{t\geq h_i} + Q_j(1 + \lfloor \frac{c(t-H_i)}{Q_{\text{tot}}} \rfloor) 1_{t\geq H_i}$, $h_i = \frac{\sum_{j\neq i} Q_j + d_j^{\max}}{c}$ and $H_i = h_i + \frac{Q_i - d_i^{\max} + \sum_{j\neq i} Q_j}{c}$. The correction term is obtained by subtracting two terms: The former, function S_j, gives the maximum possible interference caused by an interfering class j in a backlogged period of the class of interest i and is derived from a detailed analysis of DRR ; and the latter gives the effective interference caused by an interfering class j in a backlogged period of the class of interest i, given the knowledge of an arrival curve of that interfering class, α_j.

Two additional improvements are used in [90]. The former, called *line shaping*, uses the fact that, if a collection of flows is known to arrive on the same link, the rate limitation imposed by the link can be used to derive, for the aggregate flow, an arrival curve that is smaller than the sum of arrival curves of the constituent flows (as explained in Section 2.1.1.3). This improvement is also known under the name of grouping and is used, for example, in [42, 156, 157]. The other improvement, called *offsets*, uses the fact that, if several periodic flows have the same source and if their offsets are known, the temporal separation imposed by the offsets can be used to compute, for the aggregate flow, an arrival curve that is also smaller than the sum of arrival curves of the constituent flows (the latter would correspond to an adversarial choice of the offsets). Both improvements reduce the arrival curves, hence the delay bounds. Note that both improvements are independent of the correction term (and, unlike the correction term, are correct); they can be applied to any method used to compute delay bounds, as we do in Section 4.6.

4.2.3 Bouillard's Strict Service Curves

A new method to compute strict service curves for DRR that account for the interfering arrival curve constraints was recently presented in [91]; the method works only when arrival curves are concave and the aggregate strict service curve is convex, and it improves on the strict service curve of Boyer et al. Specifically, in [91, Theorem 1], for the class of interest i, there exists non-negative numbers $\hat{H}_J$ for any

$J \in \{1,\ldots,n\} \setminus \{i\}$ such that $\beta_i^{\text{Bouillard}}$ is given by

$$\beta_i^{\text{Bouillard}} = \sup_{J \subseteq \{1,\ldots,n\} \setminus \{i\}} \frac{Q_i}{\sum_{j \notin J} Q_j} \left[\beta - \sum_{j \in J} \alpha_j - \hat{H}_J \right]^+ \tag{4.2}$$

where an inductive procedure is presented to compute $\hat{H}_J$. We call these *Bouillard*'s strict service curves.

4.3 Counter Example to The Correction Term of Soni et al.

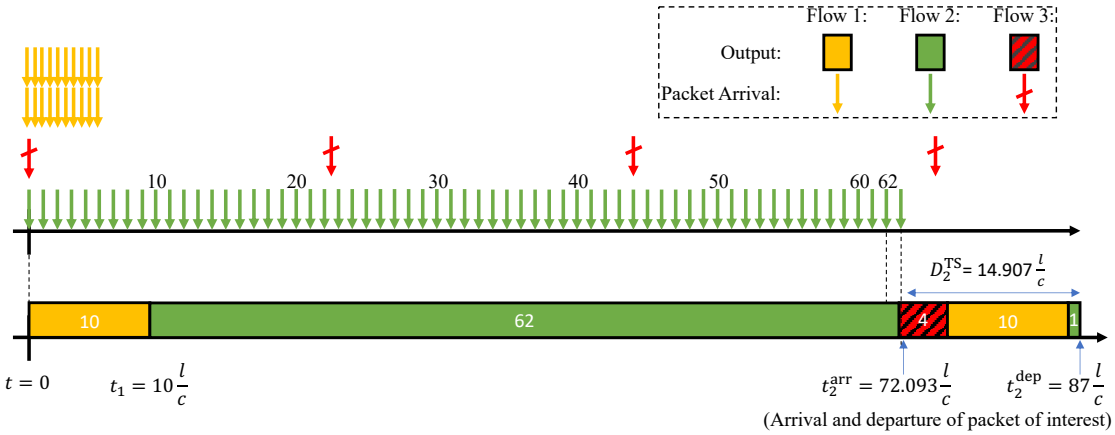


Figure 4.1: Trajectory scenario for the packet of interest of class 1 (Section 4.3.2). This packet arrives at t_2^{arr} and departs at t_2^{dep}.

In this section, we show that the delay bound of Soni et al., namely the correction term given in equation (14) in [90], rewritten using our notation in (4.1), is invalid. For class 2 in a system, we denote the delay bound of Soni et al. by $D_2^{\text{Soni-et-al}}$, and we denote the delay experienced by a packet of class 2 in the trajectory scenario by D_2^{TS}.

4.3.1 System Parameters

Consider a constant-rate server, with a rate equal to c, that uses the DRR scheduling policy. All flows have packets of constant size l, and have quanta $Q_1 = 10l$, $Q_2 = 100l$, and $Q_3 = 5l$.

Each class is constrained by a token-bucket arrival curve:

1. $\alpha_1 = \gamma_{r_1,b_1}$ with $0 \leq r_1 < \frac{Q_1}{Q_{\text{tot}}} c$ and $b_1 = 20l$.
2. $\alpha_2 = \gamma_{r_2,b_2}$ with $r_2 = 0.86c$ and $b_2 = l$.
3. $\alpha_3 = \gamma_{r_3,b_3}$ with $r_3 = 0.0401c$ and $b_3 = l$.

Assuming a token-bucket $\gamma_{r,b}$, defined in Section 2.1.1.3, for a class implies that this class has a minimum packet-arrival time equal to $\frac{l}{r}$. Also, observe that $r_i < \frac{Q_i}{Q_{\text{tot}}} c$ for $i = 1,2,3$. We compute the delay bound of Soni et al. for class 2, as explained in Section 4.2.2, and we obtain $D_2^{\text{Soni-et-al}} = 14.03383\frac{l}{c} - 1.236215\frac{\epsilon}{c}$.

4.3.2 Trajectory Scenario

We now construct a possible trajectory for our system. First, we give the inputs of our three classes. All queues are empty, and the server is idle at time $t = 0$. Then,

1. Class 1 arrives first and $A_1(t) = \min(\alpha_1(t), 20l)$ for $t > 0$ (yellow arrows in Fig. 4.1).
2. Class 2 arrives shortly after class 1 and $A_2(t) = \min(\alpha_2(t), 63l)$ for $t > 0$ (green arrows in Fig. 4.1).
3. Class 3 arrives shortly after classes 1 and 2 and $A_3(t) = \min(\alpha_3(t), 4l)$ for $t > 0$ (red arrows in Fig. 4.1).

Then, for the output, we have the following:

1) Class 1 arrives first and has 20 ready packets. As its deficit was zero before this service and $Q_1 = 10l$, the server serves 10 packets of this class. The end of the service for class 1 is $t_1 = 10\frac{l}{c}$ (the first yellow part in Fig. 4.1).

2) Then, there is an emission opportunity for class 2 and $A_2(t_1) = 9.6l$, which means class 2 has 9 ready packets at time t_1. The server starts serving packets of this class. At the end of service of these first 9 packets, at $t_2 = 19\frac{l}{c}$, class 2 has another 8 ready packets; hence, the server still serves packets of class 2. This continues and 62 packets of class 2 are served in this emission opportunity; the emission opportunity ends at $t_4 = 72\frac{l}{c}$ (the first green part in Fig. 4.1).

3) Then, there is an emission opportunity for class 3 and $A_3(t_4) = 3.8872l$, which means class 3 has 3 ready packets at time t_4. At the end of the service of 3 packets, another packet is also ready for class 3. In total, 4 packets of class 3 are served in this emission opportunity (the red part in Fig. 4.1).

4) A packet for class 2 arrives at $t_2^{\text{arr}} = 72 + \frac{0.08l}{r_1} \approx 72.093\frac{l}{c}$. This packet should wait for class 3 and class 1 to use their emission opportunities, and then it can be served. We call this *the packet of interest* of class 2, for which we capture the delay (the first blue arrow, at t_2^{arr}, on Fig. 4.1).

5) For class 1, again, 10 packets are served (the second yellow part in Fig. 4.1).

6) Finally, the packet of interest is served and its departure time is $t_2^{\text{dep}} = 87\frac{l}{c}$.

It follows that the delay for the packet of interest is $D_2^{\text{TS}} = t_2^{\text{dep}} - t_2^{\text{arr}} = 15\frac{l}{c} - \frac{0.08l}{r_2} \approx 14.907\frac{l}{c}$. Note that $D_2^{\text{TS}} > D_2^{\text{Soni-et-al}}$. To fix ideas, if $l = 100$ bytes and $c = 100$ Mb/s, the delay bounds are $D_2^{\text{Boyer-et-al}} = 146.228\mu s$, $D_2^{\text{Soni-et-al}} = 112.172\mu s$, and $D_2^{\text{TS}} = 119.256\mu s$.

4.3.3 The Contradiction with the Bound of Soni et al.

We found a trajectory scenario such that $D_2^{\text{Soni-et-al}}$ is not a valid delay bound. Let us explain why the approach of Soni et al., presented in [90], gives an invalid delay bound. In [90], it is implicitly assumed that as the delay for a packet of class 2 is upper bounded by $D_2^{\text{Boyer-et-al}}$ (the obtained delay bound using the strict service curve of Boyer et al. for class 2), only packets of interfering classes arriving within a duration $D_2^{\text{Boyer-et-al}}$ will get a chance to delay a given packet of class 2. However, in the trajectory scenario given in Section 4.3.2, all packets of class 3 (an interfering class for class 2) arriving within the time interval $[0, 75\frac{l}{c}]$ with the duration $75\frac{l}{c} >> D_2^{\text{Boyer-et-al}} = 18.3\frac{l}{c} - 2.15\frac{\epsilon}{c}$ delay the packet of interest of class 2.

Remark: Soni et al. recently have revisited the method in [158]; in (4.1), they replaced α_j, the input arrival curve for class j, by α_j^*, an output arrival curve for flow j. Thus, the delay bounds obtained by their method are less good. However, its validity remains a question mark as still, the method assumes that only packets of interfering classes arriving within a duration of a delay bound of the class of interest will get a chance to delay a given packet.

4.4 New DRR Strict Service Curve

Our next result is new DRR strict service curves that do not take into account the arrival curves of the interfering traffic. First, a non-convex strict service curve for DRR. We show that it is the largest one and that it dominates the state-of-the-art rate-latency strict service curve for DRR by Boyer et al. We also give simpler, lower approximations of it. Specifically, we also find a convex strict service curve and two rate-latency strict service curves.

Theorem 4.1 (Non-convex Strict Service Curve for DRR)**.** *Let S be a server shared by n classes that uses DRR , as explained in Section 4.1, with quantum Q_i for class i. Recall that the server offers a strict service curve β to the aggregate of the n classes. For any class i, d_i^{max} is the maximum residual deficit (defined in Section 4.2.1).*

Then, for every i, S offers to class i a strict service curve β_i^{NCDM} given by $\beta_i^{NCDM}(t) = \gamma_i\left(\beta(t)\right)$ with

$$\gamma_i(x) = \left(\lambda_1 \otimes \nu_{Q_{tot},Q_i}\right)\left(\left[x - \psi_i\left(Q_i - d_i^{max}\right)\right]^+\right) + \min\left(\left[x - \sum_{j\neq i}\left(Q_j + d_j^{max}\right)\right]^+, Q_i - d_i^{max}\right) \tag{4.3}$$

$$Q_{tot} = \sum_{j=1}^{n} Q_j \tag{4.4}$$

$$\psi_i(x) \stackrel{def}{=} x + \sum_{j,j\neq i} \phi_{i,j}(x) \tag{4.5}$$

$$\phi_{i,j}(x) \stackrel{def}{=} \left\lfloor \frac{x + d_i^{max}}{Q_i} \right\rfloor Q_j + \left(Q_j + d_j^{max}\right) \tag{4.6}$$

Here, $\nu_{p,b}$ is the stair function, λ_1 is the unit rate function and $\otimes$ is the min-plus convolution, all described in Fig. 2.3. Also we use the notation $[x]^+ = \max(x, 0)$.

Furthermore, β_i^{NCDM} is super-additive.

The proof is in Section 4.7.1. See Fig. 4.2 for some illustrations of β_i^{NCDM} (NCDM: non-convex, degraded mode). Our strict service curve captures the round-robin manner of DRR: The fact that the service is interrupted in order to serve other queues, and then the class is served at a rate given by server; each plateau (i.e., horizontal line) corresponds to a service interruption for class i. The first plateau shows the maximum service interruption where all other classes j are served by $(Q_j + d_j^{\max})$, and in the rest of their services, they are served by Q_j, as no deficit remains to carry on. Also, class i is served by $(Q_i - d_i^{\max})$ in its first service, and it is served by Q_i in other services, as it always carries the maximum residual deficit $d_i^{\max}$. Observe that γ_i in (4.3) is the strict service curve obtained when the aggregate strict service curve is $\beta = \lambda_1$ (i.e., when the aggregate is served at a constant, unit rate). In the common case where β is equal to a rate-latency function, say $\beta_{c,T}$, we have $\beta_i^{\mathrm{NCDM}}(t) = \gamma_i(c(t-T))$ for $t \geq T$ and $\beta_i^{\mathrm{NCDM}}(t) = 0$ for $t \leq T$, namely, β_i^{NCDM} is derived from γ_i by a re-scaling of the x axis and a right-shift.

The key point of Theorem 4.1 is as follows: We take into account the details of DRR in our analysis, thanks to the pseudo-inverse technique; all constraints are written without any attempts to invert them in closed form, and only at the final step they are inverted; this contrasts with previous papers (e.g., Boyer et al. in [89]) where, for tractability, approximations needed to be made at multiple inversion steps. Specifically, as shown in Section 4.7.1, in $(s,t]$, a backlogged period of the class of interest i, the service received by an interfering class j (i.e., $D_j(t)-D_j(s)$) is upper bounded as a function of the service received by class i (i.e., $\phi_{i,j}(D_i(t)-D_i(s))$, where $\phi_{i,j}$ is defined in (4.6) and illustrated in Fig. 4.3). Using $\phi_{i,j}$, as it is, results in a non-convex strict service curve of Theorem 4.1.

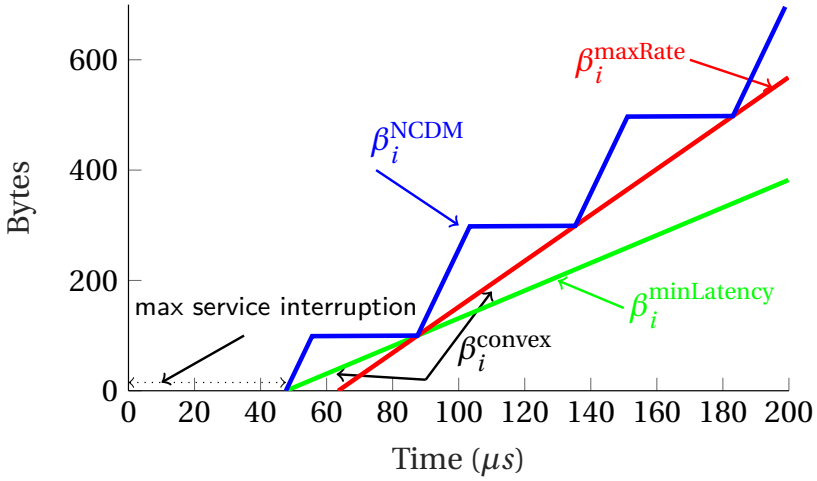


Figure 4.2: Strict service curves for DRR for an example with three input classes, quanta $=\{199,199,199\}$ bytes, maximum residual deficits $d^{\max}=\{99,99,99\}$ bytes, and $\beta(t)=ct$ with $c=100$ Mb/s (i.e., the aggregate of all classes is served at a constant rate). The figure shows the non-convex DRR strict service curve β_i^{NCDM} of Theorem 4.1; it also shows the two rate-latency strict service curves β_i^{maxRate} (same as that Boyer et al.) and $\beta_i^{\text{minLatency}}$ in Corollary 4.1 and the convex service curve $\beta_i^{\text{convex}}=\max\left(\beta_i^{\text{maxRate}},\beta_i^{\text{minLatency}}\right)$ in Corollary 4.2.

We then show that the strict service curve we have obtained is the best possible one.

Theorem 4.2 (Tightness of the DRR Strict Service Curve)**.** *Consider a DRR subsystem that uses the DRR scheduling algorithm, as defined in Section 4.1. Assume the following system parameters are fixed: the number of input classes n, the quantum Q_j allocated to every class j, maximum residual deficits d_j^{max} for every class j, and the strict service curve β for the aggregate of all classes. We assume that β is Lipschitz-continuous, i.e., there exists a constant $K>0$ such that $\frac{\beta(t)-\beta(s)}{t-s}\le K$ for all $0\le s<t$. Let i be the index of one of the classes.*

Assume that $b_i\in\mathscr{F}$ is a strict service curve for class i in any system that satisfies the specifications above. Then $b_i\le\beta_i^{NCDM}$ where β_i^{NCDM} is given in Theorem 4.1.

The proof is in Section 4.7.2. The idea of the proof is as follows: For any value of the system parameters, for any $\tau>0$, and for any class i, we create a trajectory scenario of a system such that

$$\begin{aligned}&\exists s\ge 0,\ (s,s+\tau] \text{ is backlogged for class } i\\ &\text{and } D_i(s+\tau)-D_i(s)=\beta_i^{\text{NCDM}}(\tau)\end{aligned}\tag{4.7}$$

,i.e., for the class of interest i, we create a backlogged period of duration τ where the service received by class i is exactly equal to $\beta_i^{\text{NCDM}}(\tau)$. Then, it follows that every other strict service curve b_i is upper bounded by β_i^{NCDM}. Note that assuming the aggregate strict service curve β is Lipschitz-continuous does not appear to be a restriction as the rate at which data is served has a physical limit. We then provide closed-form for the network calculus delay bounds when the class of interest i is constrained by frequent types of arrival curves, as defined in Section 2.1.1.3.

Theorem 4.3 (Closed-form Delay Bounds Obtained with the Non-convex Strict Service Curve of DRR)**.** *Make the same assumptions as in Theorem 4.1, yet with one difference: Assume that the aggregate strict service curve is a rate-latency function, i.e., $\beta = \beta_{c,T}$. Also, assume that class of interest i has $\alpha_i \in \mathscr{F}$ as an arrival curve. Let ψ_i be defined as in* (4.5)*.*

Then, the closed-form of the network calculus delay bound $hDev(\alpha_i, \beta_i^{NCDM})$ is given as follows:

1) if α_i is a token-bucket arrival curve, i.e., $\alpha_i = \gamma_{r_i,b_i}$ with $r_i \leq \frac{Q_i}{Q_{tot}}$,

$$T + \max\left(\frac{\psi_i(b_i)}{c}, \frac{\psi_i(\alpha_i(\tau_i))}{c} - \tau_i\right) \tag{4.8}$$

with $\tau_i = \frac{Q_i - (b_i + d_i^{max}) \bmod Q_i}{r_i}$.

2) if α_i is a token-bucket arrival curve and we take into account the effect of grouping, i.e., $\alpha_i(t) = \min\left(ct + l_i^{max}, \gamma_{r_i,b_i}(t)\right)$ with $r_i \leq \frac{Q_i}{Q_{tot}}$,

$$T + \max\left(\frac{\psi_i(\alpha_i(\tau_i))}{c}, \frac{\psi_i(\alpha_i(\bar{\tau}_i))}{c} - \bar{\tau}_i\right) \tag{4.9}$$

with $\tau_i = \frac{b_i - l_i^{max}}{c - r_i}$ and $\bar{\tau}_i = \frac{Q_i - (\alpha_i(\tau_i) + d_i^{max}) \bmod Q_i}{r_i}$.

3) if α_i is a stair arrival curve, i.e., $\alpha_i(t) = a_i \lceil \frac{t}{b_i} \rceil$ with $\frac{a_i}{b_i} \leq \frac{Q_i}{Q_{tot}}$,

$$T + \max\left(\frac{\psi_i(a_i)}{c}, \frac{\psi_i(\alpha_i(\tau_i))}{c} - \tau_i\right) \tag{4.10}$$

with $\tau_i = \lceil \frac{Q_i - (a_i + d_i^{max}) \bmod Q_i}{a_i} \rceil b_i$

The proof is in Section 4.7.3. The idea of the proof is that we write $\text{hDev}\left(\alpha_i, \beta_i^{\text{NCDM}}\right) = \sup_{t \geq 0} \left\{\beta_i^{\text{NCDM}\downarrow}(\alpha_i(t)) - t\right\}$, as in [38, Prop. 3.1.1], and we plug in the α_i and β_i^{NCDM}. Theorem 4.3 enables us to compute the exact delay bounds in a very simple closed form, independent of the complicated expression of our non-convex strict service curve. However, we provide simplified lower bounds of the non-convex strict service curve for DRR when analytic, closed-form expressions are important. As explained, the function $\phi_{i,j}(x)$, defined in (4.6), is the maximum interference that class j can create in any backlogged period of class i, such that class i receives a service x. Using $\phi_{i,j}$, as it is, results in the strict service curve of Theorem 4.1, which has a complex expression. If there is interest in simpler expressions, any lower bounding function is a strict service curve. In Theorem 4.4, we show that any upper bounding of function $\phi_{i,j}$, (which gives a lower bound on γ_i) results in a lower bound of β_i^{NCDM}, which is a valid, though less good, strict service curve for DRR.

Theorem 4.4 (Lower Bounds of Non-convex Strict Service Curves for DRR)**.** *Make the same assumptions as in Theorem 4.1. Also, for class i, consider functions $\phi'_{i,j} \in \mathscr{F}$ such that for every other class $j \neq i$,*

$\phi'_{i,j} \geq \phi_{i,j}$. Let ψ'_i be defined as in (4.5) *by replacing functions $\phi_{i,j}$ with $\phi'_{i,j}$ for every class $j \neq i$. Then, let γ'_i be the lower-pseudo inverse of ψ'_i, i.e., $\gamma'_i = \psi'^{\downarrow}_i$.*

Let β'_i be the result of Theorem 4.1 by replacing functions $\phi_{i,j}$, ψ_i, and γ_i with $\phi'_{i,j}$, ψ'_i, and γ'_i.

Then, S offers to class i a strict service curve β'_i and $\beta'_i \leq \beta_i^{NCDM}$.

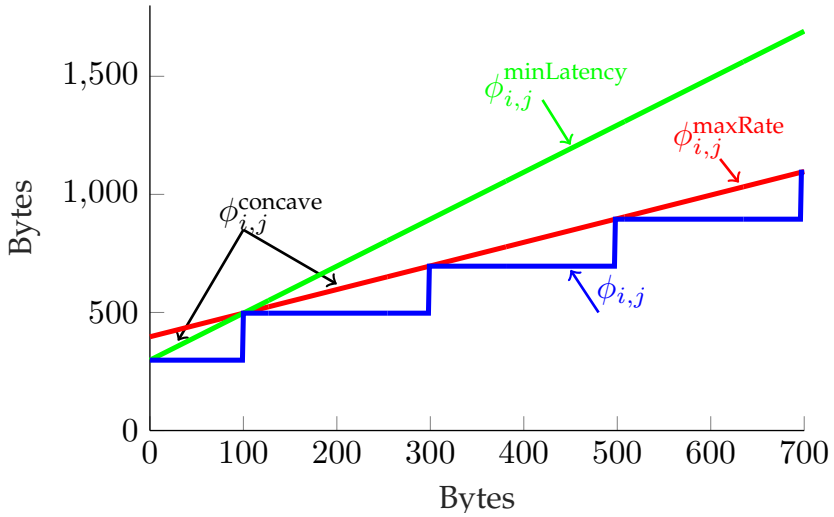


Figure 4.3: Illustration of functions $\phi_{i,j}$, $\phi_{i,j}^{\text{maxRate}}$, $\phi_{i,j}^{\text{minLatency}}$, and $\phi_{i,j}^{\text{concave}}$ defined in (4.6), (4.11), (4.12), and (4.17), respectively. These functions are obtained for the example of Fig. 4.2.

The proof is in Section 4.7.4. The idea of the proof is to show β'_i is in $\mathscr{F}$ (i.e., is wide-sense increasing) and lower bounds β_i^{NCDM}; then, the conclusion follows from the fact that any lower bound in $\mathscr{F}$ of a strict service curve is a strict service curve. There is often interest in service curves that are piecewise linear and convex; a simple case is a rate-latency function. Specifically, convex piecewise-linear functions are stable under addition and maximum, and the min-plus convolution can be computed in automatic tools very efficiently [40, Sec. 4.2]. Observe that, if the aggregate service curve β is a rate-latency function, replacing γ_i by a rate-latency (resp. convex) lower-bounding function also yields a rate-latency (resp. convex) function for β_i^{NCDM}, and vice-versa. Therefore, we are interested in rate-latency (resp. convex) functions that lower bound γ_i. We now give two lower bounds of the non-convex strict service curve for DRR using Theorem 4.4 that are common: a convex lower bound and two rate-latency lower bounds.

To obtain a rate-latency strict service curve, we use two affine upper bounds of $\phi_{i,j}$: $\phi_{i,j}^{\text{maxRate}}$, which results in a rate-latency function with the maximum rate, and $\phi_{i,j}^{\text{minLatency}}$, which results in a rate-latency function with the minimum latency (Fig. 4.3). They are defined by

$$\phi_{i,j}^{\text{maxRate}}(x) \stackrel{def}{=} \frac{Q_j}{Q_i}\left(x + d_i^{\max}\right) + Q_j + d_j^{\max} \tag{4.11}$$

$$\phi_{i,j}^{\text{minLatency}}(x) \stackrel{def}{=} \frac{Q_j}{Q_i - d_i^{\max}} x + Q_j + d_j^{\max} \tag{4.12}$$

Corollary 4.1 (Rate-Latency Strict Service Curve for DRR)**.** *With the assumption in Theorem 4.1 and the*

definitions (4.11)-(4.12), *S offers to every class i strict service curves* $\gamma_i^{maxRate}\left(\beta(t)\right)$ *and* $\gamma_i^{minLatency}\left(\beta(t)\right)$ *with*

$$\gamma_i^{maxRate} = \beta_{R_i^{max},T_i^{max}} \tag{4.13}$$

$$\gamma_i^{minLatency} = \beta_{R_i^{min},T_i^{min}} \tag{4.14}$$

$$R_i^{max} = \frac{Q_i}{Q_{tot}} \text{ and } T_i^{max} = \sum_{j,j\neq i} \phi_{i,j}^{maxRate}(0) \tag{4.15}$$

$$R_i^{min} = \frac{Q_i - d_i^{max}}{Q_{tot} - d_i^{max}} \text{ and } T_i^{min} = \sum_{j,j\neq i} \phi_{i,j}^{minLatency}(0) \tag{4.16}$$

The right-hand sides in (4.13) *and* (4.14) *are the rate-latency functions defined in Section 2.1.*

The above result is obtained by using Theorem 4.4 with $\phi_{i,j}^{\text{maxRate}}$ and $\phi_{i,j}^{\text{minLatency}}$; hence, $\gamma_i \geq \gamma_i^{\text{maxRate}}$ and $\gamma_i \geq \gamma_i^{\text{minLatency}}$. Also, observe that the strict service curve of Boyer et al., explained in Section 4.2.1, is equal to $\gamma_i^{\text{maxRate}}\left(\beta(t)\right)$. It follows that β_i^{NCDM} dominates it; hence, obtained delay bounds using β_i^{NCDM} are guaranteed to be less than or equal to those of Boyer et al.

A better upper bound on $\phi_{i,j}$ can be obtained by taking its concave closure (i.e., the smallest concave upper bound) that is equal to the minimum of $\phi_{i,j}^{\text{maxRate}}$ and $\phi_{i,j}^{\text{minLatency}}$:

$$\phi_{i,j}^{\text{concave}}(x) = \min\left(\phi_{i,j}^{\text{maxRate}}(x), \phi_{i,j}^{\text{minLatency}}(x)\right) \tag{4.17}$$

Corollary 4.2 (Convex Strict Service Curve for DRR)**.** *With the assumption in Theorem 4.1 and the definitions* (4.13)-(4.14), *S offers to every class i a strict service curve* $\gamma_i^{convex}\left(\beta(t)\right)$ *with*

$$\gamma_i^{convex}(x) = \max\left(\gamma_i^{maxRate}(x), \gamma_i^{minLatency}(x)\right) \tag{4.18}$$

The above result is obtained by using Theorem 4.4 with $\phi_{i,j}^{\text{concave}}$. Also, it can be shown that it is the largest convex lower bound of γ_i. When β is a rate-latency function, this provides a convex piecewise-linear function, which has all the good properties mentioned earlier.

4.5 New DRR strict Service Curves that Account for Arrival Curves of Interfering Classes

The next result provides a method to improve on any strict service curve by taking into account the arrival curve constraints of interfering classes. It can thus be applied to the strict service curves presented in Section 4.4 and to Bouillard's strict service curves.

4.5.1 A Mapping to Refine Strict Service Curves for DRR by Accounting for Arrival Curves of Interfering Classes

Theorem 4.5 (Non-convex, Full Mapping)**.** *Let S be a server with the assumptions in Theorem 4.1. Also, assume that every class i has an arrival curve* $\alpha_i \in \mathscr{F}$ *and a strict service curve* $\beta_i^{old} \in \mathscr{F}$, *and let* $N_i = \{1,2,\ldots,n\} \setminus \{i\}$, *and for any* $J \subseteq N_i$, *let* $\bar{J} = N_i \setminus J$.

Then, for every class i, a new strict service curve $\beta_i^{new} \in \mathscr{F}$ is given by

$$\beta_i^{new} = \max\left(\beta_i^{old}, \max_{J \subseteq N_i} \gamma_i^J \circ \left[\beta - \sum_{j \in \bar{J}} \left(\alpha_j \oslash \beta_j^{old}\right)\right]_{\uparrow}^{+}\right) \tag{4.19}$$

with

$$\gamma_i^J(x) = \left(\lambda_1 \otimes \nu_{Q_{tot}^{J,i}, Q_i}\right)\left(\left[x - \psi_i^J\left(Q_i - d_i^{max}\right)\right]^{+}\right) + \min([x - \sum_{j \in J}\left(Q_j + d_j^{max}\right)]^{+}, Q_i - d_i^{max}) \tag{4.20}$$

$$Q_{tot}^{J,i} = Q_i + \sum_{j \in J} Q_j \tag{4.21}$$

$$\psi_i^J(x) \stackrel{def}{=} x + \sum_{j \in J} \phi_{i,j}(x) \tag{4.22}$$

In (4.19), $[.]_{\uparrow}^{+}$ *is the non-decreasing and non-negative closure, defined in Section 2.1, and* $\circ$ *is the composition of functions.*

The proof is in Section 4.7.5. The essence of Theorem 4.5 is as follows. Equation (4.19) gives new strict service curves β_i^{new} for every class i; they are derived from already available strict service curves β_i^{old} and from arrival curves on the input classes α_j; thus, it enables us to improve any collection of strict service curves that are already obtained.

The key point of Theorem 4.5 is as follows: As explained in Section 4.4, in $(s,t]$, a backlogged period of the class of interest i, the service received by an interfering class j (i.e., $D_j(t) - D_j(s)$) is upper bounded as a function of the service received by class i (i.e., $\phi_{i,j}(D_i(t) - D_i(s))$, where $\phi_{i,j}$ is defined in (4.6)). Also, the service received by an interfering class j is upper bounded by the output arrival curve of class j (i.e., $(\alpha_j \oslash \beta_j^{\text{old}})(t-s)$); combining both upper bounds results in our Non-convex, Full mapping.

The computation of service curves in Theorem 4.5 and of the resulting delay bounds can be restricted to a finite horizon. Indeed, all computations in Theorem 4.5 are causal except for the min-plus deconvolution $\alpha_j \oslash \beta_j^{\text{old}}$. But, as mentioned in Section 2.1, such computation and the computation of delay bounds can be limited to $t \in [0; t^*]$ for any positive t^* such that $\alpha_j(t^*) \leq \beta_j^{\text{old}}(t^*)$ for every $m \geq 1$ and $j = 1:n$. To find such a t^*, we can use any lower bound on β_j^{old}.

We then compute $t_j^* = \inf_{s>0}\{\alpha_j(s) \leq \beta_j^{\text{old}}(s)\}$ and take, as *sufficient horizon*, $t^* = \max_j t_j^*$. The computations in Theorem 4.5 can then be limited to this horizon or any upper bound on it. The computations can be performed with a tool such as RealTime-at-Work (RTaW) [92], which uses an exact representation of functions with finite horizon, by means of rational numbers with exact arithmetic.

An iterative scheme can be obtained as follows: Theorem 4.5 can be iteratively applied, starting from any available strict service curves for all classes, and for every class, an increasing sequence of strict service curves is obtained; specifically, let β_i^0 be an initial strict service curve for every class i; then, for every integer $m \geq 1$ and every class i, define β_i^m by replacing β_j^{old} with β_j^{m-1} in (4.19):

$$\beta_i^m = \max\left(\beta_i^{m-1}, \max_{J \subseteq N_i} \gamma_i^J \circ \left[\beta - \sum_{j \in \bar{J}} \left(\alpha_j \oslash \beta_j^{m-1}\right)\right]_{\uparrow}^{+}\right) \tag{4.23}$$

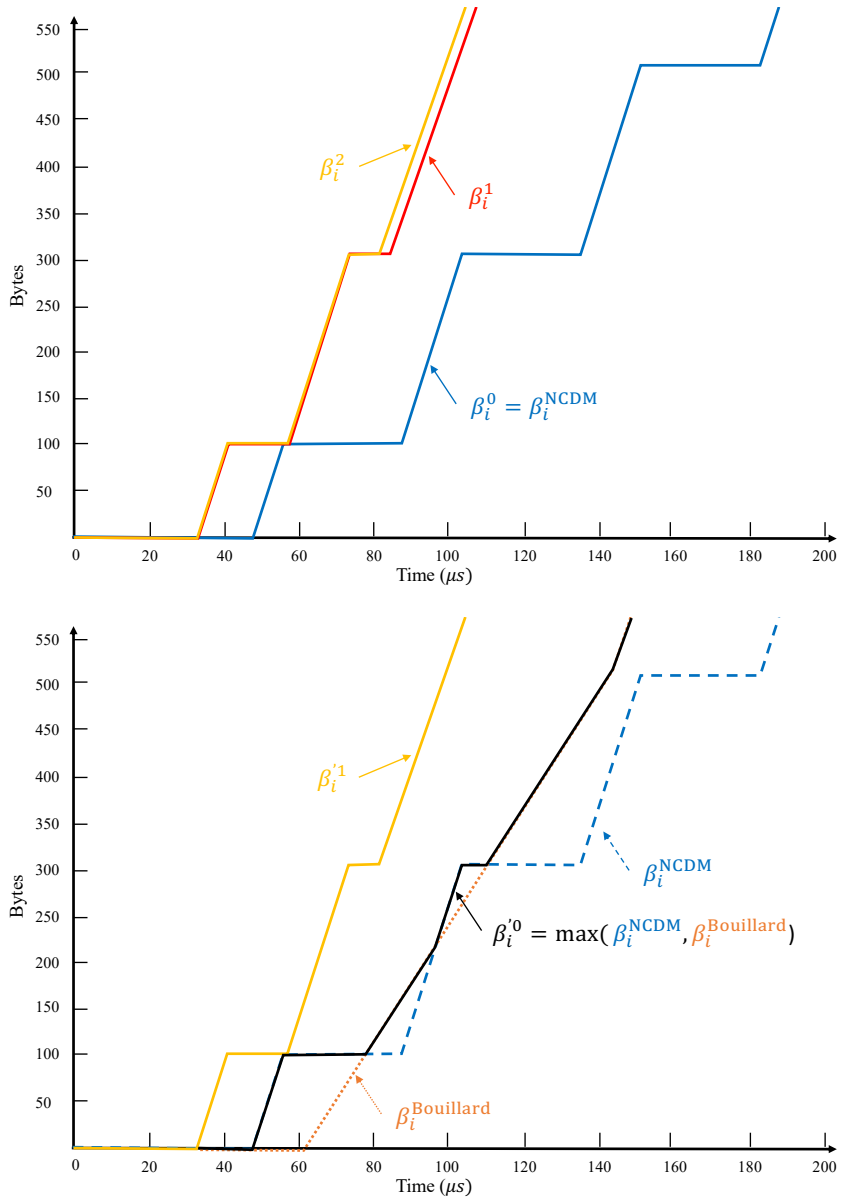


Figure 4.4: Strict service curves for class 2 of the example of Fig. 4.2, where all classes have token-bucket arrival curves with $r = \{5, 1, 1\}\frac{l}{512}$ Mb/s and $b = \{5l, l, l\}$. When iteratively applying Theorem 4.5, starting with either β_i^0 (Top: the sequence $\beta_i^0 \leq \beta_i^1 \leq \beta_i^2$) or starting with $\max(\beta_i^{\text{Bouillard}}, \beta_i^0)$ (Bottom: the sequence $\beta_i'^0 \leq \beta_i'^1 = \beta_i^2$), after 2 iterations, the strict service curves of all classes become stationary in the horizon of the figure, and the scheme stops. The sufficient horizon t^* in this example is 200μs. Obtained with the RTaW online tool.

It follows that β_i^m is a strict service curve for class i and $\beta_i^0 \leq \beta_i^1 \leq \beta_i^2 \leq \ldots$.

We are guaranteed simple convergence for the strict service curves of all classes when iteratively applying Theorem 4.5, starting from any available strict service curves for all classes. This is because, first, as explained above, computations of such strict service curves can be limited to a sufficient finite horizon; second, by iteratively applying Theorem 4.5, we obtain an increasing sequence of strict service curves for all classes, and every strict service curve is upper bounded by β, the aggregate strict service curve. In all cases that we tested, the iterative scheme became stationary in such a finite horizon. Note that the computed strict service curves at each iteration are valid, hence can be used to derive valid delay bounds; this means the iterative scheme can be stopped at any iteration. For example, the iterative scheme can be stopped when the delay bounds of all classes decrease insignificantly.

This iterative scheme can be initialized by strict service curves that do not make any assumptions on interfering traffic, as obtained in Section 4.4; specifically, recall that β_i^{NCDM} is defined in Theorem 4.1, then, for every class i, let $\beta_i^0 = \beta_i^{\text{NCDM}}$, and for every integer $m \geq 1$, β_i^m is obtained as in (4.23) (see Fig. 4.4 (top)).

Alternatively, we can first compute Bouillard's strict service curve $\beta_j^{\text{Bouillard}}$ for every class j, as explained in Section 4.2.3. Observe that $\beta_j^{\text{Bouillard}}$ does not usually dominate the non-convex service curve β_i^{NCDM} obtained in Theorem 4.1 (see Figure 4.5). Therefore, since the maximum of two strict service curves is a strict service curve, we can take the maximum of both. Specifically, define $\beta_i'^0 = \max\left(\beta_i^{\text{Bouillard}}, \beta_i^{\text{NCDM}}\right)$, then, for every integer $m \geq 1$ and every class i, define $\beta_i'^m$ as in (4.31) (see Fig. 4.4 (bottom)).

In practice, in all cases that we tested, when initializing the method with either choice, we always converge to the same strict service curve for every class (Fig. 4.4). Note that when initializing the method with strict service curves that are true for the degraded operational mode (i.e., ones that do not take into account the arrival curves of the interfering classes), the iterative scheme will always converge to the same strict service curves. This is because, in the first iteration, we take into account our strict service curve, found in Theorem 1, which is the best possible one, shown in Theorem 4.2; hence, whatever the initial strict service curves will be dominated by ours, and the scheme iterates independent from the initial strict service curve.

Observe that the computation to compute strict service curve of Theorem 4.5, β_i^{new} in (4.19), requires 2^{n-1} computations of $\gamma_i^J \circ \left[\beta - \sum_{j \in \bar{J}} \left(\alpha_j \oslash \beta_j^{\text{old}}\right)\right]_\uparrow^+$ for each J (where n is the total number of the input classes of the DRR subsystem). In some cases (class-based networks), n is small, and this is not an issue; in other scenarios (per-flow queuing), this may cause excessive complexity. To address this, we find lower bounds on the strict service curve of Theorem 4.5 where only one computation at each step m is needed; this is less costly when n is large.

Corollary 4.3 (Non-convex, Simple Mapping)**.** *Make the same assumption as in Theorem 4.5. Then, for every class i, a new strict service curve $\bar{\beta}_i^{new} \in \mathscr{F}$ is given by*

$$\bar{\beta}_i^{new} = \max\left(\beta_i^{old}, \gamma_i \circ \left(\beta + \delta_i^{old}\right)_\uparrow\right) \tag{4.24}$$

with

$$\delta_i^{old}(t) \stackrel{def}{=} \sum_{j, j \neq i} \left[\phi_{i,j}\left(\beta_i^{old}(t)\right) - \left(\alpha_j \oslash \beta_j^{old}\right)(t)\right]^+ \tag{4.25}$$

Also, $\bar{\beta}_i^{new} \leq \beta_i^{new}$.

In (4.24), $\uparrow$ *is the non-decreasing closure, defined in Table 4.2, and* $\circ$ *is the composition of functions; also, note that* β *and* δ_i^{old} *are functions of the time* t.

The proof is in Section 4.7.6. The essence of Corollary 4.3 is the same as explained after Theorem 4.5. Corollary 4.3 can be iteratively applied either starting with β_i^0, defined in Theorem 4.1, or starting with $\beta_i^{0'} = \max\left(\beta_i^{\text{Bouillard}}, \beta_i^0\right)$, i.e., the maximum of β_i^0 and the strict service curve of Bouillard, explained in Section 4.2.3 (see Fig. 4.5).

In the examples that we tested, we observed that the iterative scheme obtained with Theorem 4.5, our non-convex, full mapping, converges to the same results as the iterative scheme obtained with Corollary 4.3, our non-convex, simple mapping; however, it requires more iterations (see Fig. 4.4 and Fig. 4.5).

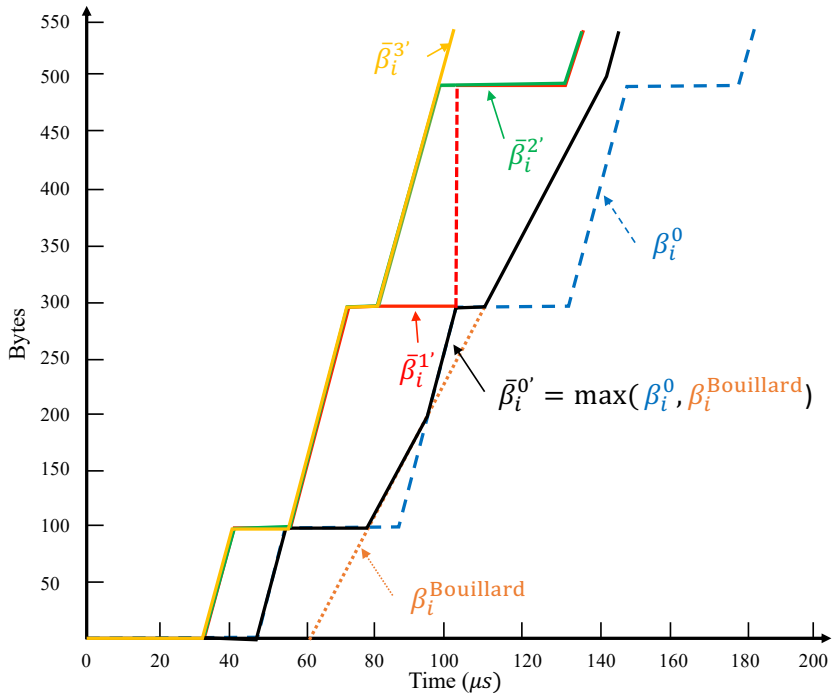


Figure 4.5: Strict service curves for flow 2 of the example of Fig. 4.4, when iteratively applying Corollary 4.3, starting with either β_i^0 (Top: the sequence $\bar{\beta}_i^0 \leq \bar{\beta}_i^1 \leq \bar{\beta}_i^2 \leq \bar{\beta}_i^3$) or starting with $\max\left(\beta_i^{\text{Bouillard}}, \beta_i^0\right)$ (Bottom: the sequence $\bar{\beta}_i^{0'} \leq \bar{\beta}_i^{1'} \leq \bar{\beta}_i^{2'} \leq \bar{\beta}_i^{3'}$), after 3 iterations, the strict service curves of all flows become stationary in the horizon of the figure, and the scheme stops. The sufficient horizon t^* in this example is 200μs. The strict service curves of the last step is precisely equal to the last step of Fig. 4.4, i.e., $\bar{\beta}_i^{3'} = \beta_i^2$. Obtained with the RTaW online tool.

4.5.2 Convex Versions of the Mapping

Computation of the strict service curves of Theorem 4.5 and Corollary 4.3 can be costly. We first explain some sources of complexity and how to address them. We then propose convex versions, for both the non-convex, full mapping in Theorem 4.5 and the non-convex, simple mapping in Corollary 4.3.

4.5.2.1 Convex Versions of Theorem 4.5

One source of complexity lies in the initial strict service curves β_i^0. For every class i, β_i^0 can be replaced by its simpler lower bounds. As presented in Section 4.4, β_i^0 can be replaced by its convex closure $\gamma_i^{\text{convex}}\big(\beta(t)\big)$, or rate-latency functions $\gamma_i^{\text{minLatency}}\big(\beta(t)\big)$ and $\gamma_i^{\text{maxRate}}\big(\beta(t)\big)$.

Another source of complexity is function γ_i^J, as defined in (4.20), is non-convex and results in strict service curves that are also non-convex (Fig. 4.4). If there is interest in simpler expressions of Theorem 4.5, any lower bounding function on γ_i^J results in a lower bound of β_i^{new}, which is a valid, though less good, strict service curve for DRR.

Corollary 4.4 (Convex, Full Mapping)**.** *Make the same assumptions as in Theorem 4.5. Also, for a class i, let $\hat{\gamma}_i^J \in \mathscr{F}$ such that $\hat{\gamma}_i^J \le \gamma_i^J$.*

Let $\hat{\beta}_i^{new}$ be the result of Theorem 4.5, in (4.19)*, by replacing functions γ_i^J with $\hat{\gamma}_i^J$.*

Then, S offers to class i a strict service curve $\hat{\beta}_i^{new}$ and $\hat{\beta}_i^{new} \le \beta_i^{new}$.

As of today, in tools such as RTaW working with functions that are linear and convex is simpler and tractable. Hence, we apply Corollary 4.4 with $\hat{\gamma}_i^J = \gamma_i^{\text{convex}J} = \max\left(\gamma_i^{\text{maxRate}J}, \gamma_i^{\text{minLatency}J}\right)$ (convex closure of function γ_i^J) and

$$\gamma_i^{\text{maxRate}J} = \beta_{R_i^{\max J}, T_i^{\max J}} \tag{4.26}$$

$$\gamma_i^{\text{minLatency}J} = \beta_{R_i^{\min J}, T_i^{\min J}} \tag{4.27}$$

$$R_i^{\max J} = \frac{Q_i}{Q_{\text{tot}}^{J,i}} \text{ and } T_i^{\max J} = \sum_{j \in J} \phi_{i,j}^{\text{maxRate}}(0) \tag{4.28}$$

$$R_i^{\min J} = \frac{Q_i - d_i^{\max}}{Q_{\text{tot}}^{J,i} - d_i^{\max}} \text{ and } T_i^{\min J} = \sum_{j \in J} \phi_{i,j}^{\text{minLatency}}(0) \tag{4.29}$$

The sequence of obtained strict service curves is thus defined by $\beta_i^{\text{convex},0} = \gamma_i^{\text{convex}} \circ \beta = \beta_i^{\text{convex}}$ and for $m \ge 1$ (see Fig. 4.6 (top)):

$$\beta_i^{\text{convex},m} = \max_{J \subseteq N_i} \gamma_i^{\text{convex}J} \circ \left[\beta - \sum_{j \in \bar{J}} \left(\alpha_j \oslash \beta_j^{\text{convex},m-1} \right) \right]_{\uparrow}^{+} \tag{4.30}$$

Alternatively, we can first compute the strict service curve of Bouillard $\beta_j^{\text{Bouillard}}$ for every class j, as explained in Section 4.2.3, and iteratively apply Corollary 4.5 with $\gamma_i^{\text{convex}J}$, starting with $\max\left(\beta_i^{\text{Bouillard}}, \beta_i^{\text{convex}}\right)$; specifically, $\beta_i^{\text{convex},0'} = \max\left(\beta_i^{\text{Bouillard}}, \beta_i^{\text{convex}}\right)$, then, for every integer $m \ge 1$ and every class i, define $\beta_i^{\text{convex},m'}$ as

$$\beta_i^{\text{convex},m'} = \max_{J \subseteq N_i} \gamma_i^{\text{convex}J} \circ \left[\beta - \sum_{j \in \bar{J}} \left(\alpha_j \oslash \beta_j^{\text{convex},m-1'} \right) \right]_{\uparrow}^{+} \tag{4.31}$$

It follows that $\beta_i^{\text{convex},m'}$ is a strict service curve for class i and at each step $m \ge 1$, one can use a better strict service curve $\max\left(\beta_i^{\text{convex},m'}, \beta_i^{\text{convex},m-1'}\right)$ (see Fig. 4.6 (bottom)).

In practice, in all cases that we tested, when initializing the method with either choice, we always

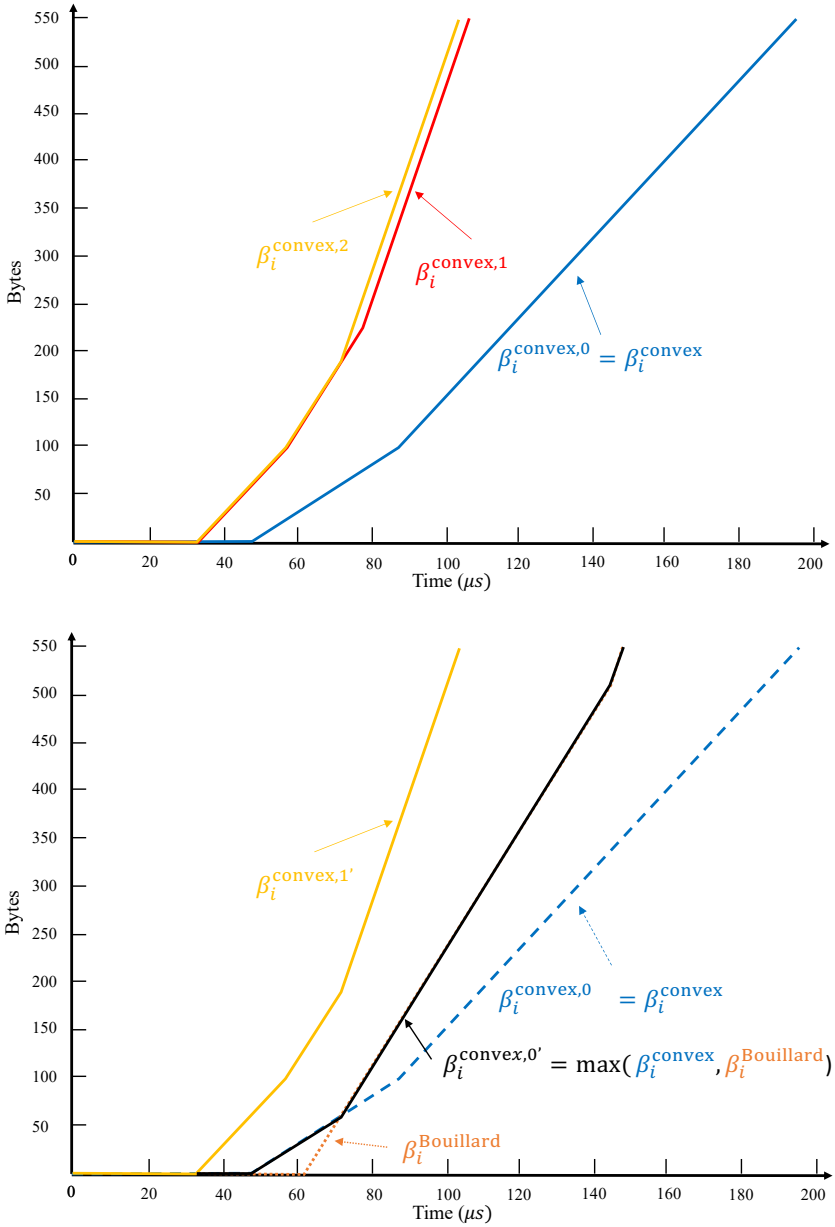


Figure 4.6: Strict service curves for class 2 of the example of Fig. 4.4, when iteratively applying Corollary 4.4 as explained in (4.30), starting with either β_i^{convex} (Top: the sequence $\beta_i^{\text{convex},0} \leq \beta_i^{\text{convex},1} \leq \beta_i^{\text{convex},2}$) or starting with $\max(\beta_i^{\text{Bouillard}}, \beta_i^{\text{convex}})$ (Bottom: the sequence $\beta_i^{\text{convex},0'} \leq \beta_i^{\text{convex},1'} = \beta_i^{\text{convex},2}$), after 2 iterations, the strict service curves of all classes become stationary in the horizon of the figure, and the scheme stops. The sufficient horizon t^* in this example is 200μ. Obtained with the RTaW online tool.

converge to the same strict service curve for every class (Fig. 4.6).

Let us explain why computing the above strict service curves is simpler. Min-plus convolution and deconvolution of piecewise linear convex can be computed in automatic tools, such as RTaW, very efficiently [40, Section 4.2]. As illustrated in Fig. 4.6, obtained strict service curves are convex, thus computing the min-plus deconvolution with such strict service curves is much simpler than with those in Fig. 4.4. Also, the composition is simpler, as for $f \in \mathscr{F}$, a function f, $\gamma_i^{\text{convex}^J}\left(f(t)\right)$ is equal to $\max\left(R_i^{\max^J}, \left[f(t) - T_i^{\max^J}\right]^+, R_i^{\min^J}\left[f(t) - T_i^{\min^J}\right]^+\right)$, which includes only multiplication, addition, and maximum operations.

4.5.2.2 Convex Versions of Corollary 4.3

Again here, a source of complexity lies in the initial strict service curves β_i^0. For every class i, β_i^0 can be replaced by its simpler lower bounds. As presented in Section 4.4, β_i^0 can be replaced by its convex closure $\gamma_i^{\text{convex}}\left(\beta(t)\right)$, or rate-latency functions $\gamma_i^{\text{minLatency}}\left(\beta(t)\right)$ and $\gamma_i^{\text{maxRate}}\left(\beta(t)\right)$.

Also, another source of complexity is function $\phi_{i,j}$ (and the resulting function γ_i). Function $\phi_{i,j}$, as defined in (4.6), is non-concave and non-linear (because it uses floor operations). This might create discontinuities that can make the computation hard, see Fig. 4.5. To address this problem, we derive the following convex version of Corollary 4.3.

Corollary 4.5 (Convex, Simple Mapping)**.** *Make the same assumptions as in Corollary 4.3. Also, for a class i, let $\phi'_{i,j}$ and γ'_i be defined as in Theorem 4.4.*

Let $\bar{\beta}_i^{new\prime}$ be the result of Corollary 4.3 by replacing functions $\phi_{i,j}$ and γ_i with $\phi'_{i,j}$ and γ'_i, respectively.

Then, S offers to every class i a strict service curve $\bar{\beta}_i^{new\prime}$.

The proof is not given in detail, as it is similar to the proof of Corollary 4.3 after replacing functions $\phi_{i,j}$ and γ_i with $\phi'_{i,j}$ and γ'_i, respectively.

We apply Corollary 4.5 as follows: Apply Corollary 4.5 by replacing $\phi_{i,j}$ and γ_i with $\phi_{i,j}^{\text{maxRate}}$ and $\gamma_i^{\text{maxRate}}$ defined in (4.11) and (4.13); also, Apply Corollary 4.5 by replacing $\phi_{i,j}$ and γ_i with $\phi_{i,j}^{\text{minLatency}}$ and $\gamma_i^{\text{minLatency}}$ defined in (4.12) and (4.14); then, we take the maximum of the two strict service curves obtained in each case.

This can be iteratively applied: In both cases, let the initial strict service curves $\beta_i^{\text{concave},0}$ be defined as in Corollary 4.2. Specifically, the sequence of obtained strict service curves are thus defined by either $\bar{\beta}_i^{\text{convex},0} = \gamma_i^{\text{convex}} \circ \beta = \beta_i^{\text{convex}}$ or $\bar{\beta}_i^{\text{convex},0} = \max\left(\beta_i^{\text{convex}}, \beta_i^{\text{Bouillard}}\right)$ and for $m \geq 1$, $\bar{\beta}_i^{\text{convex},m} = \max\left(\bar{\beta}_i^{m'}, \bar{\beta}_i^{m''}\right)$ with

$$\begin{aligned}
\bar{\beta}_i^{m'} &= \gamma_i^{\text{minLatency}} \circ \left(\beta + \delta_i^{\text{minLatency},m-1}\right)_\uparrow, \\
\bar{\beta}_i^{m''} &= \gamma_i^{\text{maxRate}} \circ \left(\beta + \delta_i^{\text{maxRate},m-1}\right)_\uparrow, \\
\delta_i^{\text{minLatency},m-1} &= \sum_{j \neq i} \left[\phi_{i,j}^{\text{minLatency}} \circ \bar{\beta}_i^{\text{convex},m-1} - \alpha_j \oslash \bar{\beta}_j^{\text{convex},m-1}\right]^+, \\
\delta_i^{\text{maxRate},m-1} &= \sum_{j \neq i} \left[\phi_{i,j}^{\text{maxRate}} \circ \bar{\beta}_i^{\text{convex},m-1} - \alpha_j \oslash \bar{\beta}_j^{\text{convex},m-1}\right]^+.
\end{aligned} \tag{4.32}$$

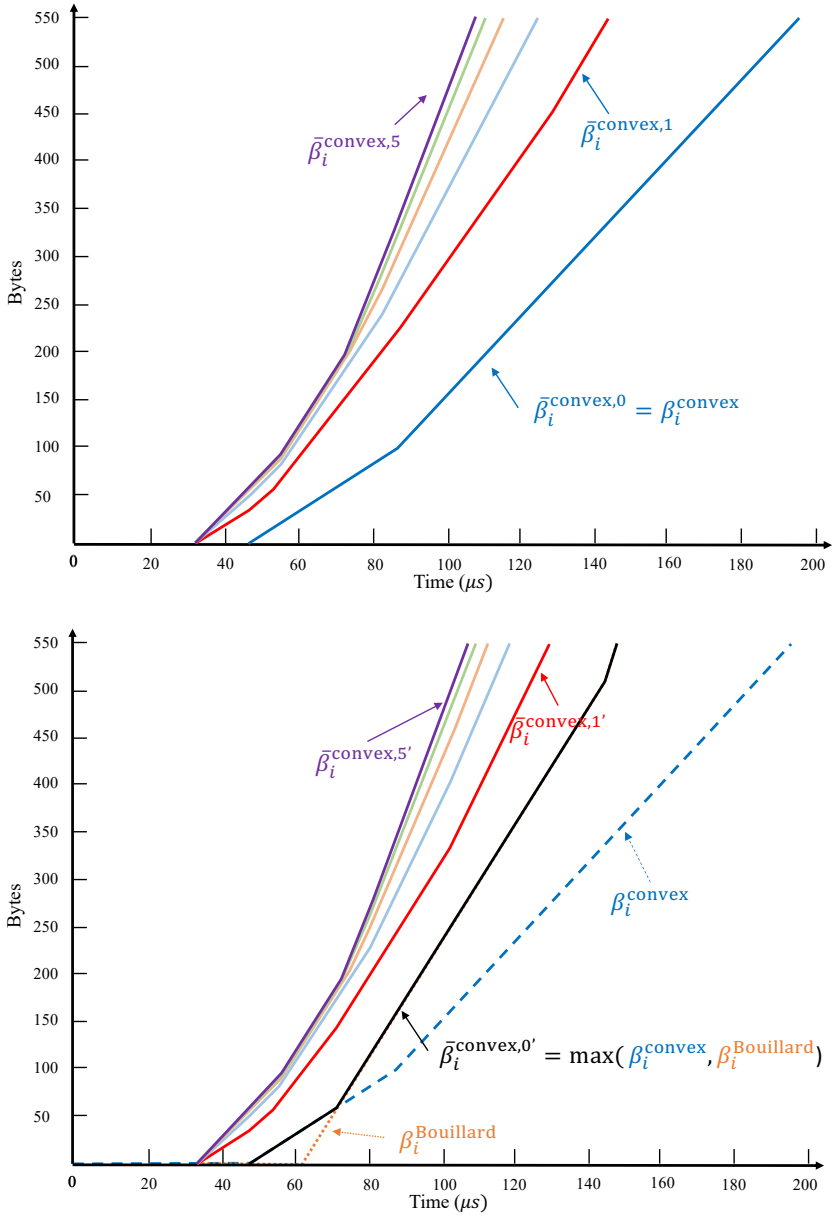


Figure 4.7: Strict service curves for class 2 of the example of Fig. 4.4, when iteratively applying Corollary 4.5, starting with either β_i^{convex} (Top: the sequence $\bar{\beta}^{\text{convex},0} \leq \bar{\beta}^{\text{convex},1} \leq \ldots$) or starting with $\max(\beta_i^{\text{Bouillard}}, \beta_i^{\text{convex}})$ (Bottom: the sequence $\bar{\beta}^{\text{convex},0'} \leq \bar{\beta}^{\text{convex},1'} \leq \ldots$). The iterative scheme stops when the computed delay bounds for all classes decrease by less than 0.25μs. The sufficient horizon t^* in this example is 200μs. The delay bounds obtained with the strict service curve of the last iteration of both cases are equal, however, the strict service curves are different. The strict service curves of all classes become stationary after 16 iterations. Obtained with the RTaW online tool.

Let us explain why computing the above strict service curves is simpler (see Fig. 4.7). The first reason is in computing the composition of $\phi_{i,j}^{\text{maxRate}}$ (resp. $\phi_{i,j}^{\text{minLatency}}$) with another function. Observe that for a function $f \in \mathscr{F}$, $\phi_{i,j}^{\text{maxRate}}\left(f(t)\right)$ (resp. $\phi_{i,j}^{\text{minLatency}}\left(f(t)\right)$) is equal to $\frac{Q_j}{Q_i} f(t) + \phi_{i,j}^{\text{maxRate}}(0)$ (resp. $\frac{Q_j}{Q_i - d_i^{\max}} f(t) + \phi_{i,j}^{\text{minLatency}}(0)$), which includes only multiplication, addition, and minimum operations. The second reason is in computing the min-plus deconvolution; min-plus convolution and deconvolution of piecewise linear convex can be computed in automatic tools, such as RTaW, very efficiently [40, Section 4.2], and as illustrated in Fig. 4.7, obtained strict service curves are convex, thus computing the min-plus deconvolution with such strict service curves is much simpler than with those in Fig. 4.5. The last reason is in computing the composition of $\gamma_i^{\text{maxRate}}$ (resp. $\gamma_i^{\text{minLatency}}$) with another function. Observe that for a function $f \in \mathscr{F}$, $\gamma_i^{\text{maxRate}}\left(f(t)\right)$ (resp. $\gamma_i^{\text{minLatency}}\left(f(t)\right)$) is equal to $R_i^{\max}\left[f(t) - T_i^{\max}\right]^+$ (resp. $R_i^{\min}\left[f(t) - T_i^{\min}\right]^+$), which again includes only multiplication, addition, and maximum operations.

Alternatively, one can apply Corollary 4.5 by replacing $\phi_{i,j}$ and γ_i with $\phi_{i,j}^{\text{concave}}$ and γ_i^{convex} defined in (4.17) and (4.18); however, in this case, there is no guarantee that this version conserves convexity and we do not consider it further.

In the examples that we tested, we observed that the iterative scheme obtained with Corollary 4.4, our convex, full mapping, converges to the same results as the iterative scheme obtained with Corollary 4.5, our convex, simple mapping; however, it requires more iterations (see Fig. 4.6 and Fig. 4.7).

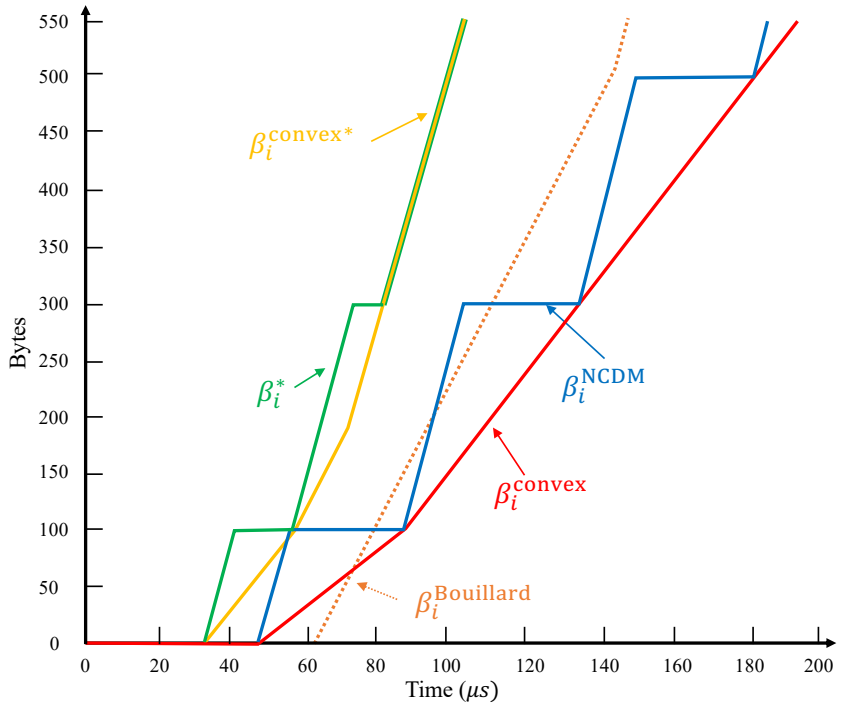


Figure 4.8: A summary of strict service curves for class 2 of the example of Fig. 4.4. The strict service curves β_i^0 and β_i^{convex} are our non-convex and convex strict service curve of Section 4.4, with no assumption on the interfering traffic. The strict service curves β_i^* and $\beta_i^{\text{convex}*}$ are our best non-convex and convex strict service curve that accounts for the interfering traffic, explained in Section 4.5. Obtained with the RTaW online tool.

4.6 Numerical Evaluation

Table 4.1: Delays bounds of all classes of Section 4.6.1.

Class	Boyer et al.	Thm. 4.1	Bouillard	Thm. 4.5	Cor. 4.4	Simulation (non-degraded)	Simulation (degraded)
Electric protection (μs)	52	44.51	52	44.51	52	44.51	44.51
Virtual reality games (ms)	1.75	1.74	1.33	1.32	1.32	1.32	1.74
Video conference (ms)	2.61	2.61	1.82	1.81	1.81	1.81	2.61
4k videos (ms)	5.78	5.77	2.74	2.72	2.72	2.71	5.77

In this section, we compare the obtained delay bounds by using our new strict service curves for DRR, presented in Sections 4.4 and 4.5, to those of Boyer et al., Bouillard, and Soni et al. We use all network configurations that were presented by Bouillard in [91] and Soni et al. in [90], specifically, one single server, two illustration networks, and an industrial-sized one. For the illustration networks, we use the exact same configuration of classes and switches that Soni et al. use. For the industrial-sized network, Soni kindly replied to our e-mail request by saying that, for confidentiality reasons, they do not have the rights to provide more details about the network configuration than what is already given in [90]. Consequently, we use the same network but randomly choose the missing information (explained in detail in Section 4.6.3).

4.6.1 Single Server

We use the exact same configuration of classes and the server that Bouillard uses in [91]. Consider a DRR subsystem with four classes of traffic: Electric protection, Virtual reality games, Video conference, and 4k videos constrained with token bucket arrival curves with bursts $b = \{42.56, 2160, 3240, 7200\}$ kb and rates $r = \{8.521, 180, 162, 180\}$ Mbps, respectively; also, the packet sizes are $l^{\max} = \{3.04, 12, 12, 12\}$ kb. The server is a constant-rate server with a rate equal to $c = 5$Gb/s, i.e., $\beta(t) = ct$. All classes have the same quantum equal to 16000 bits.

The delay bounds obtained with different methods are given in Table 4.1. Our delay bounds always improve on those of Boyer et al. and Bouillard; when the delay is very small (electric protection), our non-convex service curves bring a considerable improvement. As discussed in Section 4.5, the results are the same with the non-convex, full mapping (Theorem 4.5) and the non-convex simple mapping (Corollary 4.4). The results of the convex full and simple mappings (Corollary 4.3 and Theorem 4.5) are also identical, but less good than the former. Also, the results are the same for all choices of initial strict service curves. Finally, we used the same trajectory scenario, given in Section 4.7.2, to provide a lower bound on the worst-case delay; we observe that our delay bounds are tight (for 4k videos almost tight), i.e., exactly equal to the worst-case delays, in either degraded (i.e., when some traffic classes misbehave) and non-degraded (i.e., when arrival curve constraints can be assumed for interfering traffic) operational mode in this single server example. As we showed that the delay bounds of Soni et al. are incorrect, we do not compute them for this example, but for the sake of comparison, we will compute them for their case studies.

4.6.2 Illustration Networks

Examples 1 and 2 are illustrated in Fig. 4.9 and 4.10. We use the exact same network with the exact same configuration for flows and switches as used by Soni et al. in [90]. Examples 1 and 2 differ only by the configuration of the switch S_4. Flows $\{v_1 \dots v_5\}$, $\{v_6 \dots v_{12}\}$, and $\{v_{13} \dots v_{20}\}$ are assigned to class C_1, C_2, and C_3, respectively. There is one DRR scheduler at every switch output port and there are $n = 3$

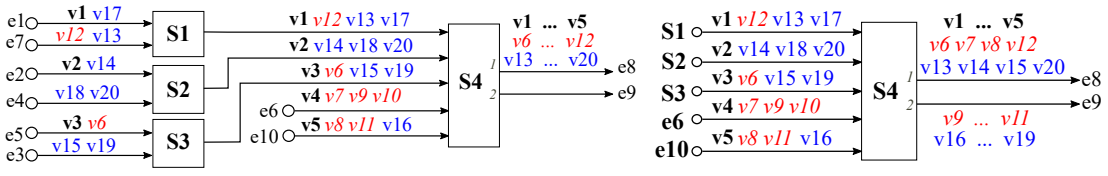


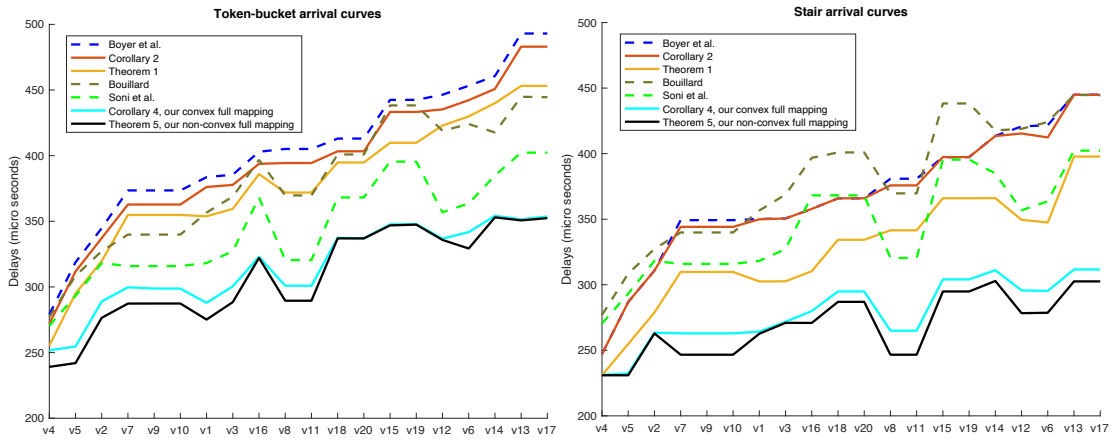
Figure 4.9: Networks of Examples 1 and 2, taken from Soni et al. [90]. Examples 1 and 2 differ only by the configuration of the switch S_4.

Flows v_i	$T_i(\mu sec)$	$l_i^{max}(byte)$
v_{12}, v_{20}	512	100
v_1, v_7, v_8, v_9, v_{17}	512	99
v_2, v_4, v_5, v_{10}, v_{13}, v_{16}, v_{18}	256	100
v_3, v_{11}, v_{14}, v_{15}, v_{19}	256	99
v_6	96	100

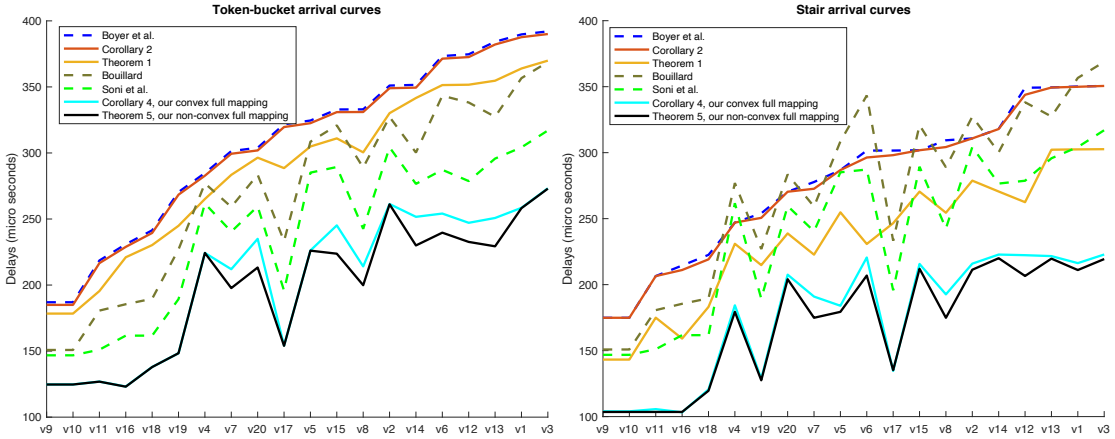
Figure 4.10: Flows Parameters for networks of Examples 1 and 2, taken from Soni et al. [90].

classes. Inside a class, arbitration is FIFO (all packets of all flows of a given class are in the same FIFO queue). Also, as in [90], we assume that queuing is on output ports only. All classes have the same quantum equal to 199 bytes. The rate of the links is equal to $c = 100$ Mb/s, and every switch S_i has a switching latency equal to 16μs. Every flow v_i has a maximum packet size $l_i^{\max}$ and minimum packet arrival T_i. Hence, flow v_i is constrained by a token-bucket arrival curve with a rate equal to $\frac{l_i^{\max}}{T_i}$ and burst equal to $l_i^{\max}$; also, it is constrained by a stair arrival curve given by $l_i^{\max}\lceil\frac{t}{T_i}\rceil$.

For the sake of comparison, as Soni et al. do not consider grouping and offsets (explained in Section 4.2.2) in these two examples, we also do not consider them. This means that the arrival curve we use for bounding the input of a class at a switch is simply equal to the sum of arrival curves expressed for every member flow. Arrival curves are propagated using the delay bounds computed at the upstream nodes. We illustrate the reported values in [90] for the delay bounds of Soni et al. For the other results, we use the RTaW online tool (Fig. 4.11). As explained in Section 2.1, RTaW provides all the necessary operations to implement our new strict service curves for DRR. First, observe that delay bounds obtained with our new strict service curves for DRR, with no knowledge of the interfering traffic, are always better than those of Boyer et al. Second, delay bounds obtained with our new strict service curve for DRR that accounts for the arrival curves of interfering flows are always better than the (incorrect) ones of Soni et al. and are considerably better than Bouillard's. The obtained delay bounds using Theorem 4.5, our non-convex full mapping, are better than or equal to those obtained using Corollary 4.4, its convex version; also, they are equal to those of Corollary 4.3, our non-convex simple mapping. Note that the results do not differ, whatever the initial strict service curves are. When using token-bucket arrival curves, the run-times (on the RTaW online tool) of Theorem 4.5 and Corollary 4.3 are in the order of 3 minutes; for their convex versions, Corollary 4.4 and Theorem 4.5, they are in the order of 30 seconds; when using stair arrival curves, the run-times (on the RTaW online tool) of Theorem 4.5 and Corollary 4.3 are in the order of 5 minutes; for their convex versions, Corollary 4.4 and Theorem 4.5, they are in the order of 1 minute and 45 seconds, respectively.



(a) Example 1



(b) Example 2

Figure 4.11: Delay bounds of flow $v_1, v_2, \ldots, v_{20}$ in Example 1 and Example 2 of Fig. 4.9. In each example, we follow [90] and assume once that flows are constrained by token-bucket arrival curves, and once that flows are constrained by stair arrival curves. The delay bounds of Soni et al. are taken from [90], and other results are computed with the RTaW online tool. First, delay bounds obtained with our new strict service curves for DRR, with no knowledge on the interfering traffic, are always better than those of Boyer et al. Second, delay bounds obtained with our new strict service curve for DRR that accounts for the arrival curve of interfering flows are always better than those of Soni et al. and are considerably better than the delay bounds of Bouillard. The obtained delay bound obtained with Theorem 4.5 and Corollary 4.4, our non-convex and convex full mapping, are equal to those obtained with Corollary 4.3 and Theorem 4.5, our non-convex and convex simple mapping. In each plot, flows are ordered by values of Boyer's bound. The state-of-the-art, i.e., delay bounds of Boyer et al., Soni et al, and Bouillard are plotted with dashed lines.

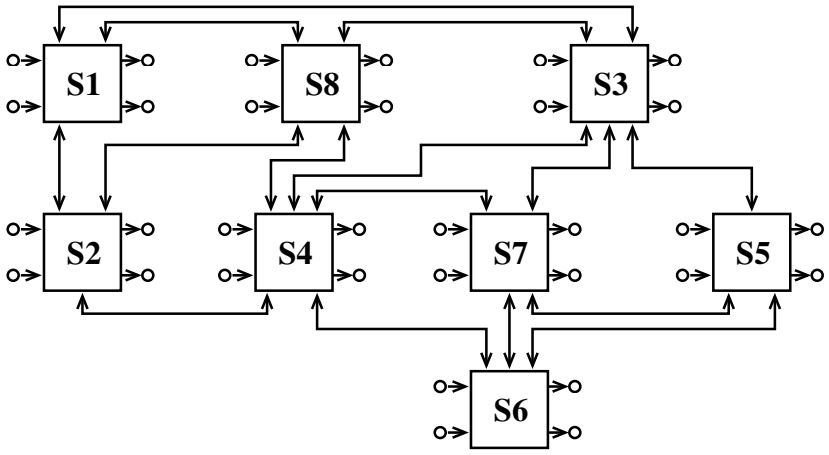


Figure 4.12: Industrial-sized network topology. The figure is taken from [31].

4.6.3 Industrial-Sized Network

We use the network of Fig. 4.12; it corresponds to a test configuration provided by Airbus in [42]. The industrial-sized case study that Soni et al. use in [90] is based on this network in [31]. We combine the available information in both papers to understand this network. It includes 96 end-systems, 8 switches, 984 flows, and 6412 possible paths. The rate of the links is equal to $c = 100$ Mb/s, and every switch S_i has a switching latency equal to 16μs. We find that each switch has 6 input and 6 output end-systems. Three classes of flows are considered: critical flows, multimedia flows, and best-effort flows. There is one DRR scheduler at every switch output port with $n = 3$ classes. At every DRR scheduler, the quanta are 3070 bytes for the critical class, 1535 bytes for the multimedia class, and 1535 bytes for the best-effort class. 128 multicast flows, with 834 destinations, are critical; they have a maximum packet size equal to 150 bytes and their minimum packet arrival time is between 4 and 128ms. 500 multicast flows, with 3845 destinations, are multimedia and their class has a quantum equal to 1535 bytes; they have a maximum packet size equal to 500 bytes; and their minimum packet-arrival time is between 2 and 128ms. 266 multicast flows, with 1733 destinations, are best-effort; they have a maximum packet-size equal to 1535 bytes; and their minimum packet arrival time is between 2 and 128ms. For every flow, the path from the source to a destination can traverse at most 4 switches. Specifically, 1797, 2787, 1537, and 291 source-destination paths have 1, 2, 3, and 4 hops, respectively. We choose the paths randomly and satisfy all these constraints.

Due to the limited expressiveness of the language used by the RTaW online tool, we could not implement the industrial-size network there. Therefore, we used MATLAB, which has the required expressiveness. The obtained delay bounds are quasi-identical for the full and simple versions of the mappings, therefore we illustrate results only for Theorem 4.5 (non-convex full mapping) and Corollary 4.4 (convex full mapping).

Note that the results are identical for both mentioned choices of initial strict service curves. We also computed the delay bounds obtained with the strict service curve of Boyer et al., with Bouillard's strict service curve and with the correction term of Soni et al. In all cases, and as in [90], the arrival curve used for bounding the input of a class at a switch incorporates the effects of delay bounds computed upstream, as well as grouping (line shaping) and offset (Section 4.2.2); furthermore, the offsets are such that they create maximum separation, as with [90]. We find that our bounds significantly improve upon

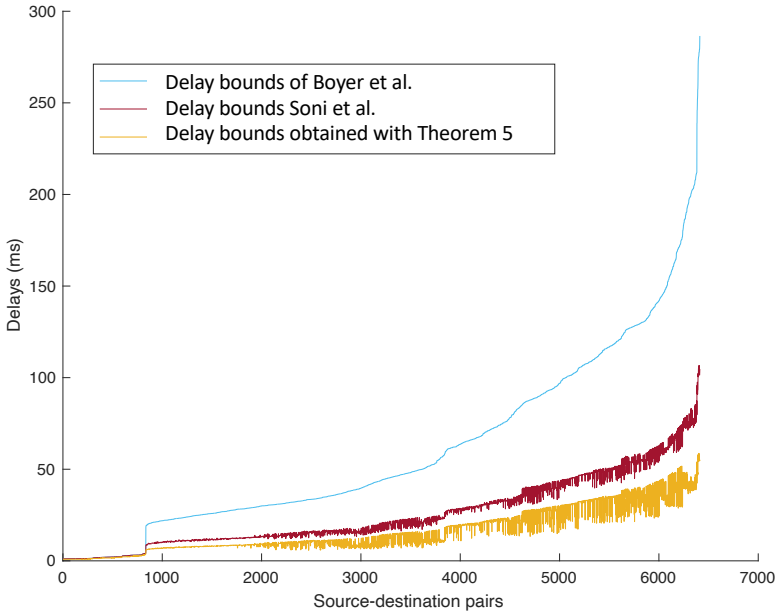


Figure 4.13: Delay bounds of the industrial case for all source-destination pairs in the system. The comparison with delay bounds with Bouillard's method is illustrated in Fig. 4.14. The obtained delay bound obtained with Theorem 4.5 and Corollary 4.4, our non-convex and convex full mapping, are equal to those obtained with Corollary 4.3 and Theorem 4.5, our non-convex and convex simple mapping. Source-destination paths are ordered by values of Boyer's bound.

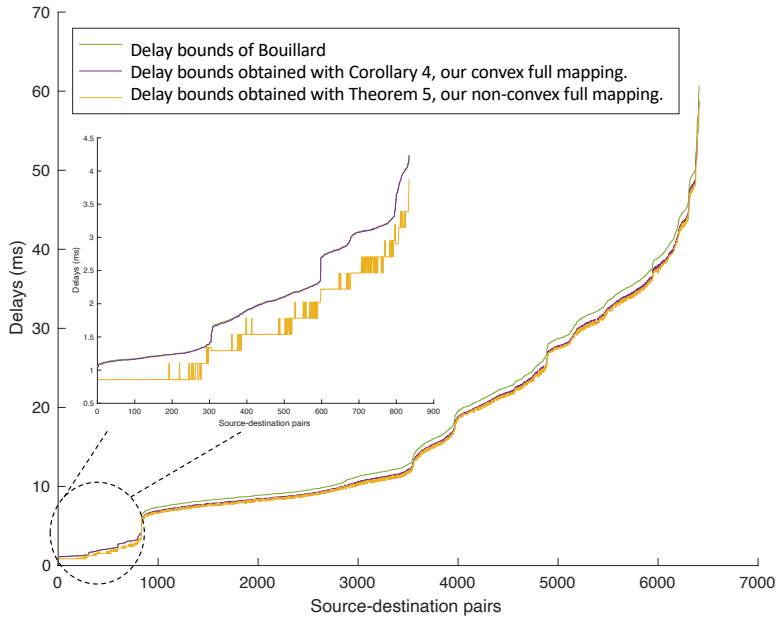


Figure 4.14: Delay bounds of the industrial case for all source-destination pairs in the system. The obtained delay bound obtained with Theorem 4.5 and Corollary 4.4, our non-convex and convex full mapping, are equal to those obtained with Corollary 4.3 and Theorem 4.5, our non-convex and convex simple mapping. Source-destination paths are ordered by values of Bouillard's bound.

the existing bounds, even the incorrect ones (Fig. 4.13). Moreover, we always improve on Bouillard's delay bounds. Also, delay bounds obtained using Theorem 4.5 are considerably improved compared to its convex version for flows with low delay bounds.

Remark on run-times: For the industrial-sized described above, run-times (on a 2.6 GHz 6-Core Intel Core i7 computer) of Theorem 4.5 and its convex version are 96 and 72 minutes, respectively; however, run-times of Corollary 4.3 and its convex version are higher and are 130 and 103 minutes, respectively. This is because the number of classes is small, i.e., 3 classes. To increase this, we divided, at uniformly random, flows of each class into three new classes, which resulted in 9 classes in total. By doing so, run-times of Theorem 4.5 and its convex version are 275 and 220 minutes, respectively; however, run-times of Corollary 4.3 and its convex version are lower and are 162 and 130 minutes, respectively. This supports the fact that the computation of strict service curves of Corollary 4.3 is faster than those of Theorem 4.5 for when the number of flows is large.

4.7 Proofs

4.7.1 Proof of Theorem 4.1

The idea of the proof is as follows. We consider a backlogged period $(s,t]$ of the class of interest i, and we let p be the number of complete service opportunities for class i in this period, where a complete service opportunity starts at line 5 and ends at line 10 of Algorithm 4.1. p is upper bounded by a function of the amount of service received by class i, given in (4.34). Given this, the amount of service received by every other class j is upper bounded by a function of the amount of service received by class i, given in (4.36). Using this result gives an implicit inequality for the total amount of service in (4.38). By using the technique of pseudo-inverse, this inequality is inverted and provides a lower bound for the amount of service received by the class of interest.

From [89, Sub-goal 1], the number p of complete service opportunities for class of interest, i, in $(s,t]$, satisfies

$$D_i(t) - D_i(s) \geq pQ_i - d_i^{\max} \tag{4.33}$$

An intuitive explanation to obtain this inequality is as follows: In this interval, class i has at least p complete services, and its residual deficit after each of these services is the maximum residual deficit, $d_i^{\max}$; thus, in its first complete service (if $p > 0$), it receives a service at least equal to $Q_i - d_i^{\max}$, and other services (if any), it receives a service at least equal to Q_i. Therefore, as p is integer:

$$p \leq \left\lfloor \frac{D_i(t) - D_i(s) + d_i^{\max}}{Q_i} \right\rfloor \tag{4.34}$$

Furthermore, it is shown in the proof of [89, Sub-goal 2] that

$$D_j(t) - D_j(s) \leq (p+1)Q_j + d_j^{\max} \tag{4.35}$$

An intuitive explanation to obtain this inequality is as follows: In this interval, class i has p complete services; it follows that all other classes j have at most $p+1$ complete service as we have a round-robin scheduler; its first service in this interval starts with its maximum residual deficit, $d_j^{\max}$, and its residual deficits remain zero after each of its services; it means in its first service it receives a service at most equal to $Q_j + d_j^{\max}$, and in all other services (if any), it receives a service at most equal to Q_j. Using (4.34),

we obtain

$$D_j(t) - D_j(s) \leq \underbrace{\left\lfloor \frac{D_i(t) - D_i(s) + d_i^{\max}}{Q_i} \right\rfloor Q_j + (Q_j + d_j^{\max})}_{\phi_{i,j}(D_i(t)-D_i(s))} \tag{4.36}$$

Next, as the interval $(s,t]$ is a backlogged period, by the definition of the strict service curve for the aggregate of classes we have

$$\beta(t-s) \leq (D_i(t) - D_i(s)) + \sum_{j \neq i} \left(D_j(t) - D_j(s)\right) \tag{4.37}$$

We upper bound the amount of service to every other class j by applying (4.36):

$$\beta(t-s) \leq \underbrace{(D_i(t) - D_i(s)) + \sum_{j, j \neq i} \phi_{i,j}\,(D_i(t) - D_i(s))}_{\psi_i(D_i(t)-D_i(s))} \tag{4.38}$$

Then we invert (4.38) using (2.13) and obtain

$$D_i(t) - D_i(s) \geq \psi_i^{\downarrow}(\beta(t-s)) \tag{4.39}$$

Lastly, we want to compute $\psi_i^{\downarrow}$. Observe that, by plugging $\phi_{i,j}$ in (4.5), $\psi_i(x) = x + \left\lfloor \frac{x+d_i^{\max}}{Q_i} \right\rfloor \left(\sum_{j \neq i} Q_j\right) +$

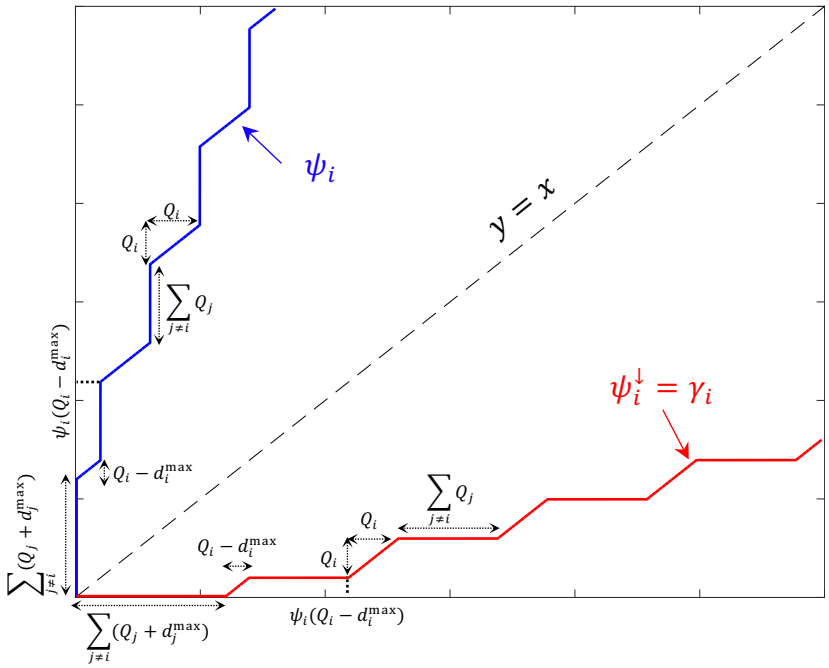


Figure 4.15: Illustration of ψ_i and its lower-pseudo inverse $\psi_i^{\downarrow}$, equal to γ_i, defined in (4.5) and (4.40), respectively. Function γ_i^J has the same form as γ_i.

$\Sigma_{j \neq i}\left(Q_j + d_j^{\max}\right)$; as there is no plateau in ψ_i, its lower-pseudo inverse is simply its inverse which is

obtained by flipping the axis (Fig. 4.15), and is obtained as

$$\psi_i^{\downarrow}(x) = \left(\lambda_1 \otimes \nu_{Q_{\text{tot}},Q_i}\right)\left(\left[x - \psi_i\left(Q_i - d_i^{\max}\right)\right]^+\right) + \min\left(\left[x - \sum_{j\neq i}\left(Q_j + d_j^{\max}\right)\right]^+, Q_i - d_i^{\max}\right) \tag{4.40}$$

$\psi_i^{\downarrow}$ is illustrated in Fig. 4.15. In (4.40), observe that the term with min expresses the finite part at the beginning between 0 and $\psi_i\left(Q_i - d_i^{\max}\right)$; also, observe that the term with the min-plus convolution expresses the rest (see Fig. 2.3.b with $a = Q_i$ and $b = Q_{\text{tot}} = \sum_{j=1}^{n} Q_j$.).

Lastly, we need to prove that β_i^{NCDM} is super-additive. This follows from the tightness result in Theorem 4.2 (the proof of which is independent of the rest of this proof). Indeed, the super-additive closure $\beta_i^{\text{NCDM}'}$ of β_i^{NCDM} is also a strict service curve, and $\beta_i^{\text{NCDM}'}(t) \geq \beta_i^{\text{NCDM}}(t)$ for all t [40, Prop. 5.6]). By Theorem 4.2, we also have $\beta_i^{\text{NCDM}'}(t) \leq \beta_i^{\text{NCDM}}(t)$ for all t, hence $\beta_i^{\text{NCDM}'}(t) = \beta_i^{\text{NCDM}}(t)$. □

4.7.2 Proof of Theorem 4.2

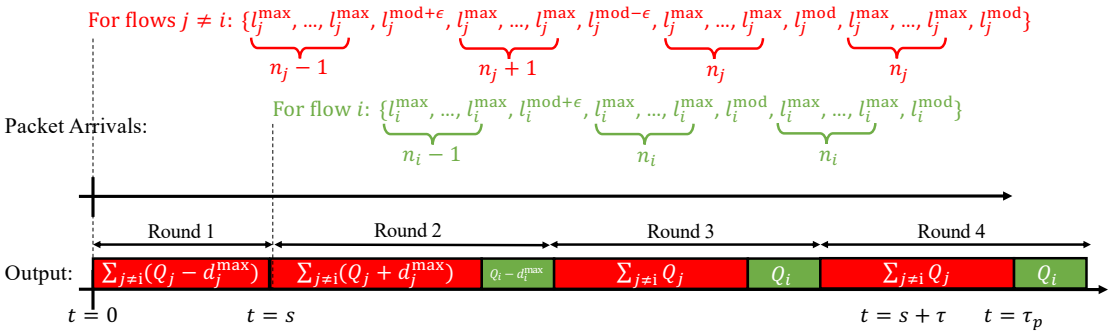


Figure 4.16: Example of the trajectory scenario presented in Section 4.7.2 with $p = 2$.

Proof of Theorem 4.2. We prove that, for any value of the system parameters, for any $\tau > 0$, and for any class i, there exists one trajectory of a system such that

$$\begin{aligned} &\exists s \geq 0,\ (s, s+\tau] \text{ is backlogged for class } i \\ &\text{and } D_i(s+\tau) - D_i(s) = \beta_i^{\text{NCDM}}(\tau) \end{aligned} \tag{4.41}$$

Step 1: Constructing the Trajectory

1) We use the following packet lengths for each class j: As explained in Section 4.1, we have $Q_j \geq l_j^{\max}$; thus, there exists an integer $n_j \geq 1$ such that $Q_j = n_j l_j^{\max} + (Q_j \bmod l_j^{\max})$. Then, let $l_j^{\text{mod}} = (Q_j \bmod l_j^{\max})$, $l_j^{\text{mod-}\epsilon} = l_j^{\text{mod}} - \epsilon$, $l_j^{\text{mod}+\epsilon} = l_j^{\text{mod}} + \epsilon$.

2) Classes are labeled in order of quanta, i.e., $Q_j \leq Q_{j+1}$.

3) At time 0, the server is idle, and the input of every queue $j \neq i$ is a bursty sequence of packets as follows:

- First, $(n_j - 1)$ packets of length $l_j^{\max}$ followed by a packet of length $l_j^{\text{mod}+\epsilon}$;
- Second, $(n_j + 1)$ packets of length $l_j^{\max}$ followed by a packet of length $l_j^{\text{mod-}\epsilon}$. Note that if $l_j^{\text{mod}} = 0$,

the sequence can be changed to n_j packets of length $l_j^{\max}$ followed by a packet of length $l_j^{\max} - \epsilon$ and the rest of the proof remains the same;

- Third, $\left(\lfloor \frac{\beta_i^{\text{NCDM}}(\tau)+d_i^{\max}}{Q_i} \rfloor - 1\right)$ times of a sequence of n_j packets of length $l_j^{\max}$ followed by a packet of length l_j^{mod}.

4) Let class i' be the first class that is visited after class i by the DRR subsystem, i.e., $i' = (i+1) \bmod n$. The input of class i' arrives shortly before all other classes $j \neq i$ at time 0.

5) The output of the system is at rate K (the Lipschitz constant of β) from time 0 to times s, which is defined as the time at which queue i is visited in the second round, namely

$$s = \frac{1}{K} \sum_{j \neq i} \left(Q_j - d_j^{\max}\right) \tag{4.42}$$

It follows that

$$\forall t \in [0, s], D(t) = Kt \tag{4.43}$$

6) The input of queue i starts just after time s, with a bursty sequence of packets as follows:

- First, $(n_i - 1)$ packets of length $l_i^{\max}$ followed by a packet of length $l_i^{\text{mod}+\epsilon}$;
- Second, $\left(\lfloor \frac{\beta_i^{\text{NCDM}}(\tau)+d_i^{\max}}{Q_i} \rfloor\right)$ times of a sequence of n_i packets of length $l_i^{\max}$ followed by a packet of length l_i^{mod}.

7) After time s, the output of the system is equal to the guaranteed service; by 3) and 6), the busy period lasts for at least τ, i.e.,

$$\forall t \in [s, s+\tau], D(t) = D(s) + \beta(t-s) \tag{4.44}$$

In particular,

$$D(s+\tau) - D(s) = \beta(\tau) \tag{4.45}$$

Step 2: Analyzing the Trajectory

Let p be the number of complete services for class i in $(s, s+\tau]$, and let τ_p be the start of the first service for class i after these p services. We want to prove that

$$D_j(\tau_p) - D_j(s) = \phi_{i,j} \left(D_i(s+\tau) - D_i(s)\right) \tag{4.46}$$

We first analyze the service received by every other class $j \neq i$. First, observe that every $j \neq i$ sends $(n_j - 1)$ packets of length $l_j^{\max}$ followed by a packet of length $l_j^{\text{mod}+\epsilon}$ in the first service after $t = 0$; this is because at the end of serving these packets, the deficit of class j becomes $d_j^{\max}$ and the head-of-the-line packet has a length $l_j^{\max} > d_j^{\max}$. Second, for the first service after time s, every other class $j \neq i$ sends $(n_j + 1)$ packets of length $l_j^{\max}$ followed by a packet of length $l_j^{\text{mod-}\epsilon}$, and at the end of this service, the deficit becomes zero. Third, observe that in any other complete services for class j (if any), it sends n_j packets of length $l_j^{\max}$ followed by a packet of length l_j^{mod}. Hence, in the first complete service of class j after time s, class j is served by $\left(Q_j + d_j^{\max}\right)$; and in every other complete service for class j, it is served by Q_j. (the red parts in Fig. 4.16)

We then analyze the service received by class i. First, it should wait for all other $j \neq i$ to use their first service after time s, and then class i sends $(n_i - 1)$ packets of length $l_i^{\max}$ followed by a packet of length $l_i^{\text{mod}+\epsilon}$; this is because at the end of serving these packets, the deficit of class i becomes $d_i^{\max}$ and the head-of-the-line packet has a length $l_i^{\max} > d_i^{\max}$. Second, observe that in any other complete services for class i (if any), it sends n_j packets of length $l_j^{\max}$ followed by a packet of length l_j^{mod}. Hence, in the first complete service of class i, which happens after time s, class i is served by $\left(Q_i - d_i^{\max}\right)$; and in every other complete service for class i, it is served by Q_i. (the green parts in Fig. 4.16)

Then, by combining the last two paragraphs, observe that (Fig. 4.16)

- Class i is served in $(s, \tau_p]$ by $\left[pQ_i - d_i^{\max}\right]^+$.
- Every other class j has $p+1$ complete services in $(s, \tau_p]$, and they are served by $\left((p+1)Q_j + d_j^{\max}\right)$.

It follows that

$$D_j(\tau_p) - D_j(s) = \phi_{i,j}\left(D_i(\tau_p) - D_i(s)\right) \tag{4.47}$$

Then, there are two cases for $s+\tau$: whether $s+\tau < \tau_p$ or $s+\tau \geq \tau_p$. In the former case, $s+\tau$ is not in the middle of a service for the class of interest, and hence $D_i(s+\tau) = D_i(\tau_p)$; in the latter case, $s+\tau$ is in the middle of a service for class i and $D_i(s+\tau) - D_i(\tau_p) < Q_i$; thus observe that $\phi_{i,j}\left(D_i(s+\tau) - D_i(s)\right) = \phi_{i,j}\left(D_i(\tau_p) - D_i(s)\right)$. Hence, in both cases, (4.46) holds.

Then, If we apply $\psi_i^{\downarrow}$ to both sides of (4.45), the right-hand side is equal to $\beta_i^{\text{NCDM}}(\tau)$. Thereby, we should prove

$$\psi_i^{\downarrow}\left(D(s+\tau) - D(s)\right) = D_i(s+\tau) - D_i(s) \tag{4.48}$$

Let $y = D(s+\tau) - D(s)$ and $x = D_i(s+\tau) - D_i(s)$. Our goal is now to prove that

$$\psi_i^{\downarrow}\left(y\right) = x \tag{4.49}$$

Again consider the two cases for $s+\tau$.

Case 1: $s+\tau < \tau_p$

In this case the scheduler is not serving class i in $[\tau_p, s+\tau]$; thus $D_i(s+\tau) = D_i(\tau_p)$. Combining it with (4.47), it follows that

$$\begin{aligned} \psi_i(x) &= x + \underbrace{\sum_{j,j\neq i} \phi_{i,j}(x)}_{\sum_{j,j\neq i}\left(D_j(\tau_p) - D_j(s)\right)} \\ y &= x + \sum_{j,j\neq i}\left(D_j(s+\tau) - D_j(s)\right) \end{aligned} \tag{4.50}$$

and thus

$$\psi_i(x) \geq y \tag{4.51}$$

Let $x - l_i^{\text{mod}} < x' < x$; class i's output becomes equal to x' during the emission of the last packet thus

$$\psi_i(x') = x' + \sum_{j,j\neq i}\left(D_j(\tau_{p-1}) - D_j(s)\right) \tag{4.52}$$

Hence

$$\forall x' \in (x - l_i^{\text{mod}}, x), \psi_i(x') < y \tag{4.53}$$

Combining (4.51) and (4.53) with Lemma 3.14 shows (4.49).

Case 2: $s+\tau \geq \tau_p$

In this case, the scheduler is serving class i in $[\tau_p, s+\tau]$. For every other class j, we have $D_j(s+\tau) = D_j(\tau_p)$. Hence, combining it with (4.46),

$$\psi_i(x) = D_i(s+\tau) - D_i(s) + \sum_{j,j\neq i} \underbrace{\phi_{i,j}\left(D_i(s+\tau) - D_i(s)\right)}_{D_j(s+\tau)-D_j(s)} = y \tag{4.54}$$

As with case 1, for any $x' \in (x - l_i^{\text{mod}}, x)$, we have $\psi_i(x) < y$, which shows (4.49).

This shows that (4.41) holds. It remains to show that the system constraints are satisfied.

Step 3: Verifying the Trajectory

We need to verify that the service offered to the aggregate satisfies the strict service curve constraint. Our trajectory has one busy period, starting at time 0 and ending at some time $T_{\max} \geq \tau$. We need to verify that

$$\forall\, t_1, t_2 \in [0, T_{\max}] \text{ with } t_1 < t_2, D(t_2) - D(t_1) \geq \beta(t_2 - t_1) \tag{4.55}$$

Case 1: $t_2 < s$

Then $D(t_2) - D(t_1) = K(t_2 - t_1)$. Observe that, by the Lipschitz continuity condition on β, for all $t \geq 0$, $\beta(t) = \beta(t) - \beta(0) = \beta(t) \leq Kt$ thus $K(t_2 - t_1) \geq \beta(t_2 - t_1)$.

Case 2: $t_1 < s \leq t_2$

Then $D(t_2) - D(t_1) = \beta(t_2 - s) + K(s - t_1)$. By the Lipschitz continuity condition:

$$\beta(t_2 - t_1) - \beta(t_2 - s) \leq K(s - t_1) \tag{4.56}$$

thus $D(t_2) - D(t_1) \geq \beta(t_2 - t_1)$.

Case 3: $s \leq t_1 < t_2$

Then $D(t_2) - D(t_1) = \beta(t_2) - \beta(t_1) \geq \beta(t_2 - t_1)$ because β is super-additive. □

4.7.3 Proof of Theorem 4.3

We first prove the following:

$$h\left(\alpha_i, \beta_i^{\text{NCDM}}\right) = T + \sup_{t\geq 0}\{\frac{1}{c}\psi_i\left(\alpha_i(t)\right) - t\} \tag{4.57}$$

First, as in [38, Prop. 3.1.1],

$$h\left(\alpha_i, \beta_i^{\text{NCDM}}\right) = \sup_{t\geq 0}\{\beta_i^{\text{NCDM}\downarrow}\left(\alpha_i(t)\right) - t\} \tag{4.58}$$

Second, we show that

$$\beta_i^{\text{NCDM}\downarrow} = T + \frac{1}{c}\gamma_i^{\downarrow} \tag{4.59}$$

As in [159, Prop. 7], for two functions $f, g \in \mathscr{F}$ where f is right-continuous, we have $(f \circ g)^{\downarrow} = g^{\downarrow} \circ f^{\downarrow}$; as $\beta_i^{\mathrm{NCDM}} = \gamma_i \circ \beta$ and γ_i is continuous , it follows that

$$\beta_i^{\mathrm{NCDM}\downarrow} = \beta^{\downarrow} \circ \gamma_i^{\downarrow} \tag{4.60}$$

Observe that $\beta^{\downarrow}(x) = \frac{1}{c}x + T$ for $x > 0$ and $\beta^{\downarrow}(x) = 0$ for $x = 0$. Combine this with the above equation to conclude (4.59).

Third, observe that $\gamma_i^{\downarrow}$ is the left-continuous version of ψ_i, and let us denote it by $\gamma_i^{\downarrow} = \psi_i^L$. It follows that

$$h\left(\alpha_i, \beta_i^{\mathrm{NCDM}}\right) = T + \sup_{t \geq 0}\{\frac{1}{c}\psi_i^L(\alpha_i(t)) - t\} \tag{4.61}$$

Observe that $\sup_{t\geq 0}\{\psi_i^L(\alpha_i(t)) - t\} = \sup_{t\geq 0}\{\psi_i(\alpha_i(t)) - t\}$ as ψ_i is right-continuous. Therefore, combining it with the above equation, (4.57) is shown.

Let us prove item 1), i.e., assuming $\alpha_i = \gamma_{r_i, b_i}$. Define H as

$$H(t) \stackrel{def}{=} T + \frac{1}{c}\psi_i(\alpha_i(t)) - t \tag{4.62}$$

Using (4.57), we have $h\left(\alpha_i, \beta_i^{\mathrm{NCDM}}\right) = \sup_{t\geq 0}\{H(t)\}$. By plugging α_i and ψ_i in H, we have $H(t) = T + \frac{\sum_{j\neq i}\left(Q_j + d_j^{\max}\right)}{c} + \frac{1}{c}\left(r_i t + b + \lfloor\frac{r_i t + b + d_i^{\max}}{Q_i}\rfloor \sum_{j\neq i} Q_j\right) - t$ Then, as $r_i \leq \frac{Q_i}{Q_{\mathrm{tot}}}c$, observe that function H is linearly decreasing between $0 \leq t < \tau_i$ with a jump at $t = \tau_i$; also, $H(t) \leq H(\tau_i)$ for all $t \geq \tau_i$ (see the above panel of Fig. 4.17). Hence, the supremum of H is obtained either at $t = 0$ or $t = \tau_i$. This concludes item 1).

Item 2) can be shown in a similar manner, however, function H is non-decreasing between $0 \leq t < \tau_i$ (see the bottom panel of Fig. 4.17).

The same proof holds for item 3). □

4.7.4 Proof of Theorem 4.4

First observe that, since $\phi'_{i,j} \in \mathscr{F}$, it follows that $\beta'_i \in \mathscr{F}$. Second, as for every $j \neq i$, $\phi_{i,j} \leq \phi'_{i,j}$, we have $\psi_i \leq \psi'_i$. In [140, Sec. 10.1], it is shown that $\forall f, g \in \mathscr{F}, f \geq g \Rightarrow f^{\downarrow} \leq g^{\downarrow}$. Applying this with $f = \psi'_i$ and $g = \psi_i$ gives that $\psi_i'^{\downarrow} \leq \psi_i^{\downarrow}$. It follows $\beta'_i(t) = \psi_i'^{\downarrow}\left(\beta(t)\right) \leq \psi_i^{\downarrow}\left(\beta(t)\right) = \beta_i(t)$. The conclusion follows from the fact that any lower bound in $\mathscr{F}$ of a strict service curve is a strict service curve. □

4.7.5 Proof of Theorem 4.5

First, we give a lemma on the operation of DRR, which follows from some of the results in the proof of Theorem 4.1.

Lemma 4.1. *Assume that the output of classes $j \in \bar{J}$ are constrained by arrival curves $\alpha_j^* \in \mathscr{F}$, i.e., D_j is constrained by α_j^*. Then, $\gamma_i^J \circ \left[\beta - \sum_{j\in\bar{J}}\alpha_j^*\right]_{\uparrow}^{+}$ is a strict service curve for class i.*

Proof. Consider a backlogged period $(s, t]$ of the class of interest i. As the interval $(s, t]$ is a backlogged

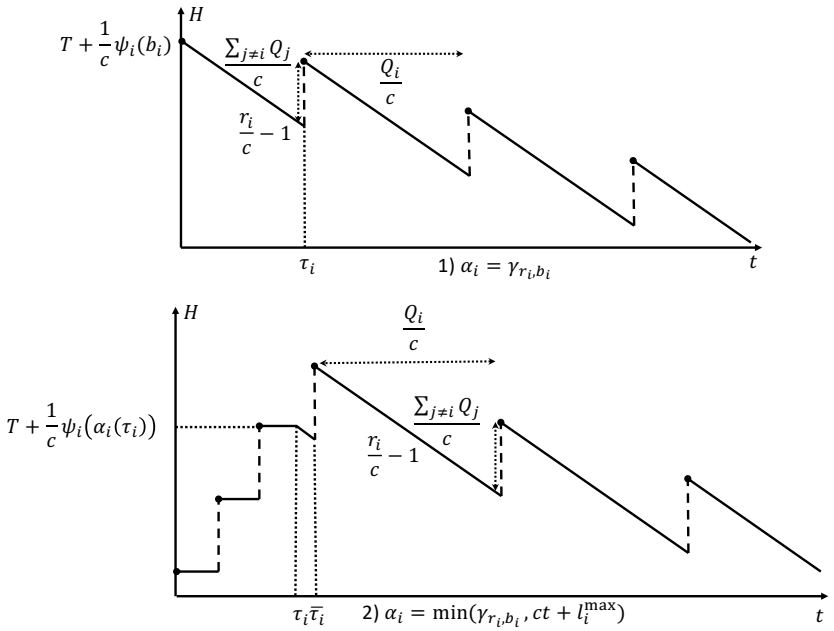


Figure 4.17: Illustration of function H defined in (4.62)

period, and since β is a strict service curve for the aggregate of flows, we have

$$\beta(t-s) \leq (D_i(t) - D_i(s)) + \sum_{j\neq i} \left(D_j(t) - D_j(s)\right) \tag{4.63}$$

For $j \in J$, upper bound $\left(D_j(t) - D_j(s)\right)$ as in (4.36), and for $j \in \bar{J}$, upper bound $\left(D_j(t) - D_j(s)\right) \leq \alpha_j^*(t-s)$, as α_j^* is an arrival curve for D_j, to obtain

$$\beta(t-s) - \sum_{j\in\bar{J}} \alpha_j^*(t-s) \leq \underbrace{(D_i(t) - D_i(s)) + \sum_{j\in J} \phi_{i,j}\,(D_i(t) - D_i(s))}_{\psi_i^J(D_i(t)-D_i(s))} \tag{4.64}$$

As ψ_i^J is a non-negative function, it follows that

$$\left[\beta - \sum_{j\in\bar{J}} \alpha_j^*\right]^+ (t-s) \leq \psi_i^J\,(D_i(t) - D_i(s)) \tag{4.65}$$

As ψ_i^J is an increasing function, it follows that the right-hand side is an increasing function of $(t-s)$. Then, by applying [40, Lemma 3.1], it follows that the inequality holds for the non-decreasing closure of the left-hand side (with respect to $t-s$), namely

$$\left[\beta - \sum_{j\in\bar{J}} \alpha_j^*\right]^+_{\uparrow} (t-s) \leq \psi_i^J\,(D_i(t) - D_i(s)) \tag{4.66}$$

Then, we use the lower pseudo-inverse technique to invert (4.66) as in (2.13),

$$D_i(t) - D_i(s) \geq \psi_i^{J\downarrow}\left(\left[\beta - \sum_{j\in\bar{J}} \alpha_j^*\right]^+_{\uparrow}(t-s)\right) \tag{4.67}$$

Hence, the right-hand side is a strict service curve for class i. Observe that $\gamma_i^J = \psi_i^{J\downarrow}$. □

We now proceed to prove Theorem 4.5. As β_j^{old} is a strict service curve for class j, it follows that $\alpha_j \oslash \beta_j^{\text{old}}$ is an arrival curve for the output of class j. Then, for every $J \subseteq N_i$, apply Lemma 4.1 with $\alpha_j^* = \alpha_j \oslash \beta_j^{\text{old}}$ for $j \in \bar{J}$ and conclude that $\gamma_i^J \circ \left[\beta - \sum_{j\in\bar{J}}\left(\alpha_j \oslash \beta_j^{\text{old}}\right)\right]^+_{\uparrow}$ is a strict service curve for class i. Lastly, the maximum over all J is also a strict service curve for class i. □

4.7.6 Proof of Corollary 4.3

We proceed with the proof by showing that for every class i, $\bar{\beta}_i^{\text{new}} \leq \beta_i^{\text{new}}$. Fix $i \leq n$ and $t \geq 0$. We want to show that $\bar{\beta}_i^{\text{new}}(t) \leq \beta_i^{\text{new}}(t)$. Let J^* be $\{j \in N_i \mid \phi_{i,j}\left(\beta_i^{\text{old}}(t)\right) < \left(\alpha_j \oslash \beta_j^{\text{old}}\right)(t)\}$. Then, observe that

$$\beta_i^{\text{new}}(t) \geq \gamma_i^{J^*}\left(\beta(t) - \sum_{j\in\bar{J}^*}\left(\alpha_j \oslash \beta_j\right)(t)\right) \tag{4.68}$$

Apply $\psi_i^{J^*}$ to the both side and observe that

$$\psi_i^{J^*}\left(\beta_i^{\text{new}}(t)\right) \geq \beta(t) - \sum_{j\in\bar{J}^*}\left(\alpha_j \oslash \beta_i^{\text{new}}\right)(t) \tag{4.69}$$

as $\beta_i^{\text{new}} \geq \beta_i^{\text{old}}(t)$ and $\phi_{i,j}$ is increasing, it follows that

$$\sum_{j\in\bar{J}^*}\phi_{i,j}\left(\beta_i^{\text{new}}(t)\right) \geq \sum_{j\in\bar{J}^*}\phi_{i,j}\left(\beta_i^{\text{old}}(t)\right) \tag{4.70}$$

Then, sum the both side of (4.69) and (4.70) to obtain

$$\psi_i\left(\beta_i^{\text{new}}(t)\right) \geq \beta(t) + \underbrace{\sum_{j\in\bar{J}^*}\left(\phi_{i,j}\left(\beta_i(t)\right) - \left(\alpha_j \oslash \beta_j^{\text{old}}\right)(t)\right)}_{\sum_{j\neq i}\left[\phi_{i,j}(\beta_i(t)) - (\alpha_j \oslash \beta_j)(t)\right]^+} \tag{4.71}$$

Then, by applying [40, Lemma 3.1], it follows that the above inequality holds for the non-decreasing closure of the left-hand side. Thus,

$$\psi_i\left(\beta_i^{\text{new}}(t)\right) \geq \left(\beta + \sum_{j\neq i}\left[\phi_{i,j}\circ\beta_i - \left(\alpha_j \oslash \beta_j^{\text{old}}\right)\right]^+\right)_{\uparrow}(t) \tag{4.72}$$

Lastly, we use the lower pseudo-inverse technique to invert as in (2.13) and as $\psi_i^{\downarrow} = \gamma_i$

$$\beta_i^{\text{new}}(t) \geq \underbrace{\gamma_i \circ \left(\beta + \sum_{j \neq i} \left[\phi_{i,j} \circ \beta_i - \left(\alpha_j \oslash \beta_j^{\text{old}} \right) \right]^+ \right)_{\uparrow} (t)}_{\tilde{\beta}_i^{\text{new}}(t)} \tag{4.73}$$

which concludes the proof. □

4.8 Conclusion

The method of the pseudo-inverse enables us to perform a detailed analysis of DRR and to obtain strict service curves that significantly improve the previous results. Our results use the network calculus approach and are mathematically proven, unlike some previous delay bounds that we have proved to be incorrect. Our method assumes that the aggregate service provided to the DRR subsystem is modeled with a strict service curve. Therefore it can be recursively applied to hierarchical DRR schedulers as found, for instance, with class-based queuing.

4.9 Notation

Table 4.2: Notation List, Specific to Chapter 4

i	A class
A	Aggregate, cumulative arrival function of all classes
D	Aggregate, cumulative departure function of all classes
α_i	An arrival curve for class i
β_i	A strict service curve offered to class i
β	A strict service curve offered to aggregate of all classes
A_i	Cumulative arrival function of class i
D_i	Cumulative departure function of class i
n	Number of classes
$l_i^{\max}$	Maximum packet size for flows of class i
$d_i^{\max}$	Maximum residual deficit of class i
Q_i	Quantum of class i
Q_{tot}	$\sum_j Q_j$
λ_c	Rate function with $\lambda_c(t) = ct$
$\beta_{R,L}$	Rate-latency function with $\beta_{c,L}(t) = \max(0, c(t-L))$
$\mathbb{R}^+$	Set of non-negative real numbers
$\mathscr{F}$	Set of wide-sense increasing functions $f : \mathbb{R}^+ \mapsto \mathbb{R}^+ \cup \{+\infty\}$
$\nu_{p,b}$	Stair function with $\nu_{p,b}(t) = b\left\lceil \frac{t}{p} \right\rceil$
$\gamma_{r,b}$	Token-bucket function with $\gamma_{r,b}(0) = 0$ and $\gamma_{r,b}(t) = rt + b$ for $t > 0$
$[x]^+$	$[x]^+ = \max(0, x)$
$\circ$	Composition of functions
$\lfloor x \rfloor$	Flooring function
hDev	Horizontal deviation $\text{hDev}(\alpha, \beta) = \sup_{t \geq 0}\{\inf\{d \geq 0 \mid \alpha(t) \leq \beta(t+d)\}\}$
$f^{\downarrow}$	Lower pseudo inverse $f^{\downarrow} = \inf\{x \mid f(x) \geq y\} = \sup\{x \mid f(x) < y\}$
$\otimes$	Min-plus convolution $(f \otimes g)(t) = \inf_{0 \leq s \leq t}\{f(t-s) + g(s)\}$
$\oslash$	Min-plus deconvolution $(f \oslash g)(t) = \sup_{s \geq 0}\{f(t+s) - g(s)\}$
$f_{\uparrow}$	Non-decreasing closure of function f defined by $\sup_{s \leq t} f(s)$
$[f]_{\uparrow}^{+}$	The non-decreasing and non-negative closure defined by $\sup_{s \leq t}[f(s)]^+$

5 Worse-Case Delay Analysis of Time-Sensitive Networks with Deficit Round-Robin

In time-sensitive networks, where delays reside,
DRR stands tall with bounds to confide.
Combining TFA and service curves strict,
Worst-case analysis, a method so slick.

Tabatabaee's characterization, the best-known service curves,
But PLP, a gem in bounds it hides.
Polynomial-size LP, a powerful tool,
Improving stability, breaking limits' rule.

Adapting PLP to DRR, an essential task,
Burstiness bounds calculated, no question to ask.
Non-convex curves, supported with care,
DRR's potential, now fully aware.

Cyclic dependencies, a challenge untold,
DRR's curves intertwined, a loop to behold.
Iterative methods, the solution we seek,
Combining the cuts, in harmony we peek.

PLP-DRR, the method we propose,
Sequential or parallel, convergence it knows.
Valid bounds, even before full accord,
At convergence, equal bounds assured.

Time-sensitive networks, with DRR in sight,
General topology, shining so bright.
Industrial application, a test so grand,
Improvements found, state-of-the-art in hand.

So let this chapter begin, a journey profound,
Exploring DRR's bounds, forever renowned.

Created with ChatGPT, free research preview (version May 24) [141]

As explained in Section 1.2.3, finding end-to-end delay bounds in a DRR network involves two steps: a single node analysis and a combination of nodes in a per-class network analysis. For the former, in the

previous chapter, we derived the best known worst-case delay bounds [86, 87, 88, 89, 91] by means of strict service curves for DRR, with or without taking into account the interference of competing DRR classes. We call this method the *DRR strict service curve*.

For the latter step, per-class network analysis, Total Flow Analysis (TFA) [40] was used in the previous chapter. TFA obtains delay bounds in FIFO networks and can be applied to per-class networks that are FIFO per class and where a service curve is known for every class at every node. When applying TFA to DRR networks, DRR strict service curves require the knowledge of burstiness bounds of competing classes inside the network, which is an output of the network analysis of TFA. Conversely, TFA needs to know the strict service curves. In the previous chapter, we solve this problem by considering only feed-forward networks (in the application example, we constrain flow routes to avoid cyclic dependencies). However, cyclic dependencies are frequent in time-sensitive networks and cannot be ignored. Recent versions of TFA [64, 119] apply to networks with cyclic dependencies and can therefore be used: For a side contribution, in Section 5.5.1, we show how to apply TFA to DRR networks with cyclic dependencies, by developing and proving the validity of, an iterative procedure, called TFA-DRR.

Our main contribution, however, goes well beyond TFA-DRR. Indeed, it is known that TFA is outperformed by Polynomial-size Linear Programming (PLP) [68] that always provides delay bounds better than or equal to those of TFA and, at high network utilization, often converges when TFA does not. Other methods, such as LUDB [22] and flow prolongations [67], also tend to dominate TFA; but, unlike PLP, they do not apply to generic topologies. This motivates the purpose of this chapter, which is to design how PLP can be applied to DRR networks. The existing PLP, like the recent versions of TFA, applies to FIFO networks with any topology. PLP consists of three phases: First, per-node delay bounds are computed (from TFA). Second, cuts are performed on the network topology in order to obtain a collection of trees and valid burstiness bounds are computed at the cuts by solving a linear program. And third, delay bounds for the flows of interest are computed on the cut network by solving another linear program for every flow of interest. PLP uses the per-node delay bounds obtained by TFA as a constraint in all its linear programs; hence it follows that the PLP delay bounds are guaranteed to be as good as the bounds obtained with TFA. An intriguing feature is that this enables PLP to obtain end-to-end delay bounds that are generally better than with TFA, whereas not using the per-node delay bounds as constraints may provide worse results.

Using PLP to analyze DRR networks requires introducing the DRR strict service curve into the PLP procedure. PLP uses internal variables such as the burstiness bounds at cuts and the per-node delay bounds, the computation of which depends on the DRR strict service curves; the DRR strict service curves depend on burstiness bounds of interfering flows at the output of every node, which can be obtained by adding to PLP another family of linear programs; the outputs of such linear programs depend on the burstiness at cuts, the per-node delay bounds, and the DRR strict service curves. In total, there are four collections of variables (burstiness bounds at cuts, burstiness bounds for interfering flows at DRR nodes, per-node delay bounds and DRR strict service curves), and the computation of every collection depends on the values of the other collections. It is natural to propose an iterative procedure as we mentioned above for the application of TFA to DRR (where there were only two collections, per-node delay bounds and DRR strict service curves), however, it is not clear how the iterations should be combined and whether some specific combinations provide better bounds. To solve this issue, we propose a generic method to combine updates to any item in the four collections in any arbitrary order, by using a distributed, shared-memory computing model. We show that the resulting bounds do not depend on how the item updates are executed, as long as every update is executed infinitely often in a hypothetical execution of infinite duration (Theorem 5.4). Still, some concrete implementations of the

method may have better execution times, and we propose two such concrete, parallel implementation methods that we apply to the industrial network used in the previous chapter.

When applying PLP to DRR, we make two further improvements. First, the existing PLP obtains burstiness bounds for individual flows, whereas the DRR strict service curve uses burstiness bounds for the aggregate of all flows for every interfering DRR class at node output. Of course, a burstiness bound for an aggregate can be obtained by summing the burstiness bounds of every individual flow, but this is generally sub-optimal. In Theorem 5.1, we extend the PLP methodology to obtain such per-aggregate bounds. Second, PLP requires convex service curves; we provide, in the previous chapter, both convex and non-convex DRR strict service curves, and the latter may obtain smaller delay bounds when the delay bounds are very small. We solve this issue with a modification of PLP, called iPLP, that adds one binary variable to the linear program per DRR node (Section 5.4.2).

The contributions of this chapter are as follows:

- We provide a method (PLP-DRR), for the worst-case timing analysis of per-class DRR network with or without cyclic dependencies, which combines DRR strict service curves and PLP in a novel way. It has three phases: (i) initial, which obtains initial TFA bounds; (ii) refinement, which improves the four collections of burstiness bounds at cuts, burstiness bounds for interfering flows at DRR nodes, per-node delay bounds, and DRR strict service curves; (iii) post-process, which obtains delay bounds for flows of interest using iPLP.

- The refinement phase uses a distributed computing model with shared memory, where individual improvements can be applied in any order. We show that any execution provides the same bounds, regardless of the order in which the individual improvements are applied. We prove that the bounds are valid. The bounds are guaranteed to be at least as good as those obtained with TFA.

- We develop, and show the validity of, a method (TFA-DRR) to apply TFA to DRR networks with cyclic dependencies. This method is of independent interest and is also used in the initial phase.

- We design two improvements to the PLP methodology. The former computes improved burstiness bounds for aggregates of flows and is used in the refinement phase. The latter enables us to use non-convex service curves in PLP and is used in the post-process phase.

- We design two concrete implementations of the method, with parallel for-loops in the refinement phase, and we apply them to the industrial network in Chapter 4. We find that the delay bounds are significantly better than the state-of-the-art.

The rest of this chapter is organized as follows. In Section 5.1, we describe the system model, including DRR operation, the network under study and the resulting graphs. In Section 5.2, we give the necessary background on DRR strict service curve, TFA, and PLP. In Section 5.3, we give a global view of PLP-DRR, our method for combining the DRR strict service curve and PLP. It uses two improvements of PLP, which are described and proven in Section 5.4. The details of PLP-DRR are described in Section 5.5, including statements about the validity and the uniqueness of the obtained bounds. In Section 5.7, we present the proofs of theorems. In Section 5.6, we apply the method to the industrial network of the previous chapter and illustrate the obtained improvements on delay bounds. In Section 5.8, we conclude the chapter. A summary of notation and symbols used in this chapter are given in Section 5.9.

5.1 System Model

We are interested in computing end-to-end delay bounds of flows in an asynchronous, time-sensitive packet-switched network with DRR.

5.1.1 Deficit Round-Robin Scheduling

The DRR subsystem and DRR algorithm are explained in Section 4.1, which we use with the following notations and a change: For each queue c, we use the notation Q_c for the assigned quantum. Also, we assumed that the DRR subsystem is placed in a larger system and can compete with other queuing subsystems, and the service offered to the DRR subsystem is modeled by means of a strict service curve $B()$. In this chapter, we assume that B is the rate-latency function with rate R and latency T, defined by $\beta_{R,T}(t) = R[t-T]^+$, unlike Section 4.1 where B is generic.

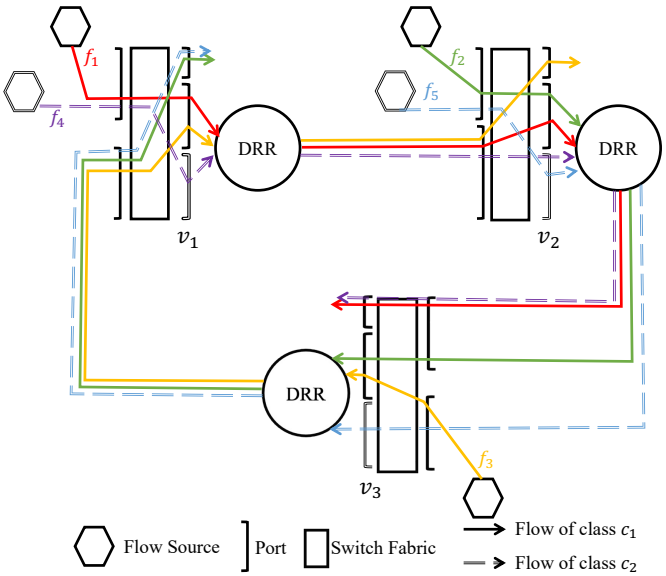
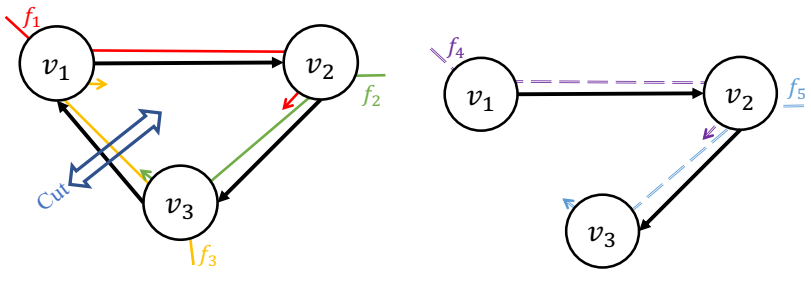


Figure 5.1: Toy Network with 2 DRR classes. Flows f_1, f_2, and f_3 belong to class c_1; flow f_4 and f_5 belong to class c_2.



(a) $\mathcal{G}_{c_1}$ (b) $\mathcal{G}_{c_2}$

Figure 5.2: The graphs induced by flows of class c_1 and c_2 of toy network of Fig. 5.1, also showing the flows path. $\mathcal{G}_{c_1}$ has one cyclic dependency, $\mathcal{G}_{c_2}$ has none.

5.1.2 Network Model and Resulting Graphs

5.1.2.1 Device Model

Devices represent switches or routers and consist of input ports, output ports, and a switching fabric. Each packet enters a device via an input port and is stored in a packetizer. A packetizer releases a packet only when the entire packet is received. Then, the packet goes through a switching fabric. A switching fabric transmits the packet to a specific output port, based on the static route of the packet. Then, the packet, based on its static class, is either queued in a FIFO-per-class queue or exits the network via a terminal port. At each non-terminal output port, packets of flows of different classes are processed according to DRR scheduling, as explained in 5.1.1. See Fig. 5.1. The aggregate service received by the DRR scheduler of a non-terminal output port $v \in \mathcal{V}$ is modeled by a strict service curve B^v, that we assume to be a rate-latency function β_{R^v,T^v} with rate R^v and a latency T^v (not to be confused with the strict service curve offered by the DRR scheduler to each class).

5.1.2.2 Flow Model

We assume that there are n classes of traffic $1,\dots,n$ in the system, and flows are statically assigned a class and a path. Traffic generated by flows is constrained at the source by means of a token-bucket arrival curve γ_{r_f,b_f} (see Section 2.1.1.3 for the definition of token-bucket arrival curves).

5.1.2.3 Graph induced by flows

For every class c, the graph $\mathcal{G}_c = (\mathcal{V}_c, \mathcal{E}_c)$ induced by flows is the directed graph defined as follows: 1) $\mathcal{V}_c \subseteq \mathcal{V}$ is the subset of all non-terminal output ports used by at least one flow of class c. 2) The directed edge $e = (v,u) \in \mathcal{E}_c$ exists if there is at least one flow of class c that traverses v and u in this order. We say that $\mathcal{G}_c$ has a cyclic dependency if it contains at least one cycle. Let $E_c^{\text{cut}} \subseteq \mathcal{E}_c$ be a cut such that artificially removing the edges in E_c^{cut} creates a tree or a forest (i.e., a collection of non-connected trees). Such cuts can be obtained by any traversal graph algorithm [160]. We keep the same node naming of vertices across graphs of different classes, namely, output ports of different classes that are connected to the same DRR scheduler have the same name. Let $\text{In}_c(v) \subset \mathcal{E}_c$ (resp. $\text{Out}_c(v)$) denote the set of edges of class c that are incidents at (resp. leave) node v. Consider the toy network of Fig. 5.1. We assume we have two classes where f_1, f_2, and f_3 belong to class c_1 and, f_4 and f_5 belong to class c_2. The graph induced by flows of c_1 and c_2, as well as the flow paths, are illustrated in Fig. 5.2. Graph $\mathcal{G}_{c_1}$ has a cycle; the figure shows one possible artificial cut to create a tree.

5.2 Background and Related Works

In this section, we provide the necessary background for analyzing a DRR system. In the network calculus framework, the classical method for this analysis combines two techniques: 1) the computation of strict service curves for each DRR class at each node, presented in Section 5.2.1; 2) the analysis of one FIFO network per DRR class. Two methods are presented, TFA in Section 5.2.2 and PLP in Section 5.2.3.

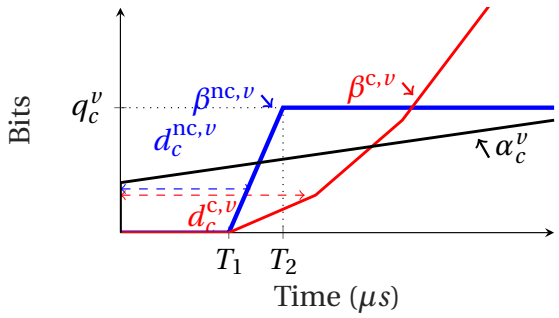


Figure 5.3: A non-convex part of a service curve $\beta^{nc,v}(t) = \min\left(\beta_{R^v,T_1}(t), q_c^v\right)$ and $T_2 = T_1 + \frac{q_c^v}{R^v}$ at some node v and for some class c. The convex function $\beta^{c,v}$ is also a valid service curve, and so is the maximum of $\beta^{c,v}$ and $\beta^{nc,v}$. The figure also shows an arrival curve α_c^v and illustrates that the bound obtained when considering the non-convex service curve, $d_c^{nc,v}$, is better than with the convex service curve, $d_c^{c,v}$, when the delay bound is small.

5.2.1 Strict Service Curves of DRR

Strict Service Curves of DRR are presented in detail in Chapter 4. In this section, we provide a notation in the specific case where arrival curves are token-bucket and the aggregate service curve is rate-latency.

5.2.1.1 Degraded Operational Mode

Let v be a node shared by n classes that uses DRR, as explained in Section 5.1.1, with quantum Q_c for class c. The node offers a strict service curve B^v to the aggregate of the n classes. Then, for every class c, node v offers to class c a strict service curve $\beta_c^{CDM,v}$ that is the maximum of two rate-latency functions, and hence, is piece-wise and convex; The rate and latency depend on the quanta and maximum residual deficits. See Appendix 5.A.1 for more details. Non-convex strict service curves of DRR can improve delay bounds when they are small, specifically if the service for a flow finishes in the first round (i.e., the flow is never backlogged at the end of each of its round of service). This motivates us to consider only the first non-convex part of the DRR strict service curve, say $\beta_c^{nc,v}$, as it corresponds to the first service round. We have $\beta_c^{nc,v}(t) = \min\left(\beta_{R^v,T_1}(t), q_c^v\right)$ where T_1 is the maximum period of time during which no data of class c can be served and (see Fig. 5.3); the exact values are given in Appendix 5.A.1. We use it as follows in Section 5.4.2: since $\beta_c^{nc,v}$ is a strict service curve, we can replace any other strict service curve for class c, β_c^v, by $\max(\beta_c^v, \beta_c^{nc,v})$, which is also a strict service curve.

5.2.1.2 Non-Degraded Operational Mode

When some arrival curves can be assumed for the interfering classes, the service received by the class of interest can be improved. We present in Chapter 4 a method that starts from service curves with no assumption on the interfering traffic (i.e., $\beta_c^{CDM,v}$ explained in Section 5.2.1.1), and iteratively improves them by taking into account the arrival curve constraints of interfering traffic.

We call $\text{DRRservice}_v^{\text{inputArrival}}(\alpha^v)$ the method that computes a collection of strict service curves for each class given α^v, input arrival curves of all classes at node v (See Appendix 5.A.2 for more details.).

We also use (5.15) in Appendix 5.A.2 (same as Lemma 4.1 in Chapter 4), which improves the strict service curves of DRR, given *output* arrival curves of every class. Specifically, we apply it when every

class c has a token-bucket arrival curve at the output, say γ_{r_c,b_c^v}, and a known strict service curve β_c^v. We call DRRservice$_v^{\text{outputBurst}}\left(\beta^v, b^v\right)$ the function that implements (5.15) in Appendix 5.A.2 and returns an improved collection of strict service curves for all classes at node v. In the above, β^v and b^v are the collection of β_c^v strict service curves and the collection of b_c^v output burstiness of all classes at node v.

In the common case where the aggregate strict service curve is a rate-latency function say β_{c_v,T_v}, the obtained strict service curves are the maximum of a finite number of rate-latency functions. Specifically, consider class c and partition other classes into two arbitrary sets J and $\bar{J}$. Then, let $r^{v,\bar{J}} = \sum_{c' \in \bar{J}} r_{c'}$ and $b^{v,\bar{J}} = \sum_{c' \in \bar{J}} b_{c'}$, i.e., $r^{v,\bar{J}}$ and $b^{v,\bar{J}}$ are the aggregated arrival rate and aggregated output burstiness bound of interfering classes in $\bar{J}$. Then, two rate-latency functions $\beta_{\bar{R}_c^{\max,v,J},\bar{T}_c^{\max,v,J}}$ and $\beta_{\bar{R}_c^{\min,v,J},\bar{T}_c^{\min,v,J}}$ can be computed for class c with

$$\bar{R}_c^{\max,v,J} = (c_v - r^{v,\bar{J}}) R_c^{\max} \tag{5.1}$$

$$\bar{R}_c^{\min,v,J} = (c_v - r^{v,\bar{J}}) R_c^{\min} \tag{5.2}$$

$$\bar{T}_c^{\max,v,J} = \frac{c_v T_v + T_c^{\max} + b^{v,\bar{J}}}{(c_v - r^{v,\bar{J}})} \tag{5.3}$$

$$\bar{T}_c^{\min,v,J} = \frac{c_v T_v + T_c^{\min} + b^{v,\bar{J}}}{(c_v - r^{v,\bar{J}})} \tag{5.4}$$

In the above, values of $R_c^{\max}$, $T_c^{\max}$, $R_c^{\min}$, and $T_c^{\min}$ depend on the quantum and the maximum residual deficit of class c and interfering classes in J; the exact values are given in (5.17)-(5.18) in Appendix 5.A.2. Thus, any choices of sets J and $\bar{J}$ results in two rate-latency functions for class c; function DRRservice$_v^{\text{outputBurst}}\left(\beta^v, b^v\right)$ computes the maximum of rate-latency functions obtained by all possible choices of sets J and $\bar{J}$. Observe that the latencies of the above functions, $\bar{T}_c^{\max,v,J}$ and $\bar{T}_c^{\min,v,J}$ in (5.3) and (5.4), are linear functions of output burstinesses of interfering classes b_c^v, and the rates, $\bar{R}_c^{\max,v,J}$ and $\bar{R}_c^{\min,v,J}$ in (5.1) and (5.2), only depend on arrival rates r_c.

5.2.2 Total Flow Analysis (TFA)

As explained in Section 2.1.3.1, in a FIFO-per-class network where a service curve is known for every class at every node, one instance of TFA is run per class, and it outputs per-node delay bounds as well as propagated burstiness for flows. If the graph induced by flows is feed-forward (i.e, cycle-free), for each node in a topological order, a delay bound and output burstiness bounds of flows are computed: the output burstiness bounds at a node are used as input by its successors in the induced graph. Else if the graph induced by flows has cyclic dependencies, no topological order can be defined and a fixed point must be computed, using an iterative method [64]. If the iteration converges to a finite value for all delay and burstiness bounds, then the network is stable and the computed bounds are valid. Otherwise, TFA diverges and the network might be truly unstable or not.

All versions of TFA (specifically, FP-TFA, SyncTFA, AsyncTFA, and AltTFA) are equivalent, i.e., they give the same bounds and stability regions [64]. We let $(d_c, z_c) = \text{GenericTFA}_c\left(\beta_c\right)$ denote any version of TFA that computes per-node delay bounds and bounds on propagated burstiness for class c, given per-node strict service curves β_c. We always apply TFA to the original (uncut) graph.

In networks with cyclic dependencies, there is a two-way dependency between TFA and DRR strict service curves: TFA needs to know DRR strict service curves a priori, however, DRR strict service curves depend on burstiness bounds of flows at the output of a node, which is a result of TFA. Specifically, on

the one hand, DRR strict service curves of node v depend on b^v, the collection of b_c^v output burstiness bound of all classes at node v (see Section 5.2.1.2). On the other hand, b^v is obtained from bounds on the propagated burstiness bounds z_c which is computed by TFA (i.e., $(d_c, z_c) = \text{GenericTFA}_c\left(\beta_c\right)$) and requires knowing β_c, the collection of DRR service curve for class c at every node v. Authors in [108] avoid this two-way dependency between TFA and DRR by restricting their analysis only to feed-forward networks.

As of today, for networks with cyclic dependencies, the only TFA solution is to use DRR strict service curves in the degraded operational mode (see Section 5.2.1.1), which only depend on the assigned quantum and maximum packet size of every class and hence can be computed for all classes at all nodes a priori to TFA; thus, per-class networks are independent and can be analyzed separately (i.e., the network is sliced into some per-class networks), and one instance of TFA can be run per-class to obtain bounds; we call this method *TFA-SOA*, and consider it the state-of-the-art as it is the straight-forward application of TFA with our DRR strict service curves.

As a first step, we propose an iterative method in Algorithm 5.1, called TFA-DRR, and prove its validity in Theorem 5.3; it combines DRR strict service curve in non-degraded operational mode (i.e., where some arrival curves can be assumed for the interfering traffic) and TFA, and it serves as the initial phase of our main method; see Fig. 5.8 for some numerical application.

5.2.3 Polynomial-size Linear Programming (PLP)

As explained in Section 2.1.3.6, PLP computes end-to-end delay bounds in FIFO networks [68], so one instance of PLP is run per class. It requires piece-wise linear convex service curves. PLP improves the bounds and stability region compared to TFA while remaining tractable. The definition of linear programs is straightforward for tree topologies. The analysis of general topologies requires first making some cuts in the induced graph in order to create a forest (i.e., one or several non-connected trees). The analysis has three steps (see Fig. 5.4):

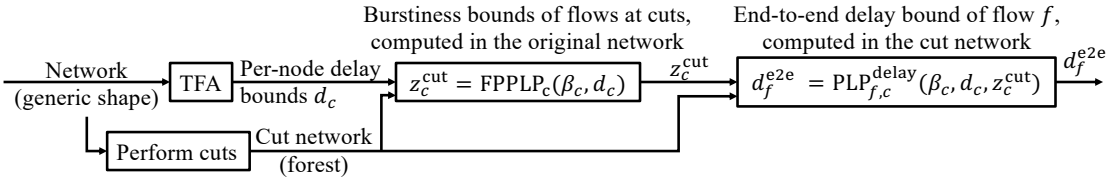


Figure 5.4: Overview of PLP analysis for the FIFO-per-class network of some class c (see items 1)-3) in Section 5.2.3).

1. TFA analysis to obtain per-node delay bounds d_c;
2. Then, output burstiness bounds at the cuts are computed. by solving one single linear program, which is equivalent to computing a fixpoint. We call $\text{FP-PLP}_c(\beta_c, d_c)$ the algorithm that computes burstiness bounds of flows at cuts given strict service curves and per-node delay bounds of class c;
3. One linear program per flow of interest obtains an end-to-end delay or backlog bound; we call $\text{PLP}_{f,c}^{\text{delay}}(\beta_c, d_c, z_c^{\text{cut}})$ (resp. $\text{PLP}_{f,c}^{\text{backlog}}(\beta_c, d_c, z_c^{\text{cut}})$) the algorithm that computes the end-to-end delay (resp. backlog) bound of flow f belonging to class c given output burstiness bounds at the cuts, strict service curves and per-node delay bounds of class c.

We use FP-PLP as is and use improved versions of $\text{PLP}^{\text{delay}}$ and $\text{PLP}^{\text{backlog}}$, as explained in Section 5.4. As PLP uses per-node delay bounds computed by TFA, the end-to-end bounds are always better than with TFA. In a network with cyclic dependencies, it is possible that TFA diverges and hence the per-node delay bounds be infinite. In this case, the constraints used by PLP that involve infinite per-node delay bounds are simply always satisfied and PLP might or might not compute finite end-to-end bounds. In general, though, PLP finds a larger stability region than TFA, i.e., it often finds finite delay bounds when the TFA per-node delay bounds are infinite.

Detailed background on these linear programs is presented in Section 5.B.

Combining PLP and DRR strict service curves require more adaptation compared to TFA: Similar to TFA, there is a two-way dependency between DRR strict service curves and $\text{PLP}^{\text{backlog}}$. Specifically, on the one hand, DRR strict service curves of node v depend on b^v, the collection of b_c^v output burstiness bound of all classes at node v (see Section 5.2.1.2). On the other hand, b^v is obtained using $\text{PLP}^{\text{backlog}}$, which requires knowing β_c (see item 3) in the above), the collection of DRR service curve for class c at every node v. Thus, this creates a level of iteration between collection b and collection β. Also, $\text{PLP}^{\text{backlog}}$ uses the collection of per-node delay bounds d (see item 3) in the above) which depends on both DRR strict service curves β and burstiness bounds of flows at the input of nodes, obtained using $\text{PLP}^{\text{backlog}}$; this imposes another level of iteration. Moreover, $\text{PLP}^{\text{backlog}}$ requires cuts and bounds on the burstiness of flows at cuts, z^{cut} (see item 3) in the above), is obtained using FP-PLP where FP-PLP requires knowing DRR strict service curves β and per-node delay bounds d (see item 2) in the above); this imposes yet another level of iteration. Thus, we have a collection of DRR service curves β, a collection of per-node delay bounds d, a collection of output bound for flows at cuts z^{cut}, and a collection of burstiness bounds b at the input and output of nodes, and we have different functions such as DRR strict service curves, $\text{PLP}^{\text{backlog}}$, FP-PLP, etc. that each uses some values of these collections and improves some other values, hence, imposing different levels of iteration; it is not clear how to combine them.

As of today, the only PLP solution is to use DRR strict service curves in the degraded operational mode (see Section 5.2.1.1), which only depend on the assigned quantum and maximum packet size of every class and hence can be computed for all classes at all nodes a priori to PLP; thus, per-class networks are independent and can be analyzed separately (i.e., the network is sliced into some per-class networks), and one instance of PLP can be run per-class to obtain bounds; we call this method *PLP-SOA*, consider it the state-of-the-art as it is the straightforward application of PLP with our DRR strict service curves. see Fig. 5.8.

We first perform the necessary adaptation of PLP to DRR by computing burstiness bounds per-class and per-output aggregate and by enabling PLP to support non-convex service curves. We then propose a generic method in Section 5.5.2.1, called PLP-DRR, for combining all these iterations sequentially and in parallel. We show, in Theorem 5.4, that obtained bounds using our method are always valid even before convergence. Also, we show that, at convergence, the bounds are the same regardless of how iterations are combined. Lastly, we present two concrete implementations, using a distributed computing model with shared memory (see Fig. 5.5 and Fig. 5.6).

5.3 Overview of the Proposed Method: PLP-DRR

Our method, called “PLP-DRR", applies the PLP methodology to DRR and is illustrated in Fig. 5.5. The starting point is the collection of per-class graphs and a cutset. We use the following notations:

- $\beta = (\beta_c^v)_{1\leq c\leq n, v\in\mathcal{V}}$, is a valid collection of strict service curves of each class c at each node v,
- $d = (d_c^v)_{1\leq c\leq n, v\in\mathcal{V}}$, is a valid collection of per-node delay bounds of each class c at each node v,
- z^{cut}, is a valid collection of output burst bounds for each flow at every edge of the cutset,
- z^{cut}, is a valid collection of output burst bounds for each flow at every edge of the cutset,
- $b = (b_c^g)_{1\leq c\leq n, g\in\mathcal{V}\cup\mathcal{E}}$ is a valid collection of burstiness bounds for aggregates of flows, indexed by a class c and by some g. The index g can be either an edge, in which case the aggregate is the set of all flows of class c carried on this edge, or a vertex v, in which case it is the set of all flows of class c that exit the output buffer represented by v.

All components of β are finite; the components of d, z^{cut} and b might be infinite.

As PLP requires per-node delay bounds, the method starts with an initial phase that performs a TFA analysis of the original (uncut) network. Note that DRR requires the service curve collection β to be computed from output burstiness bounds, which we derive from the TFA analysis; hence, we apply an iterative procedure, which we prove to be valid. At the end of this initial phase, we have a collection of service curves β and per-node delay bounds d, from which some propagated burstiness bounds (i.e., burstiness bounds for all flows at every output), hence z^{cut} and b, can be derived.

The next phase in the classical PLP methodology, FP-PLP, computes a fixpoint z^{cut} by solving a linear program. Here, however, a new value of z^{cut} allows to compute better output burstiness bounds in b using another linear program with PLP$^{\text{backlog}}$, which, in turn, allows to compute better service curves β using the DRR service curve method recalled in Section 5.2.1. The linear program in PLP$^{\text{backlog}}$ uses per-node delay bounds d, which can, in turn, be improved by re-running TFA whenever β is improved, and then PLP$^{\text{backlog}}$ could also be re-run. Also, better β and d enable FP-PLP to re-compute a better fixpoint z^{cut}, which can, in turn, be used to improve all other variables. Therefore, the second phase of the PLP methodology needs to apply a number of refinements again and again. Instead of proposing a specific arrangement of the refinements, we propose to perform them in any arbitrary order, using a shared-memory model (Fig. 5.5). As we show in Section 5.5.2, all refinements provide valid bounds, therefore, the method can be stopped at any time. However, we show that it converges to bounds (some of them possibly infinite) that are independent of the arrangement of the refinements.

The “DRR strict service curve" block in the refinement phase uses as input some burstiness bounds for the aggregate of all flows of all interfering classes at the output of a node. Such a bound could be obtained by using the existing version of PLP$^{\text{backlog}}$ applied to all flows in the aggregate. We improve both the obtained bound and the computing time by using the modification of PLP$^{\text{backlog}}$ described in Section 5.4.1.

The third phase of PLP is to obtain end-to-end bounds by applying one instance of PLP$^{\text{delay}}$ to every flow of interest. Here, we use PLP$^{\text{delay}}$ with one improvement (iPLP), which enables us to use non-convex service curves at the expense of adding a few binary variables to the linear program (Section 5.4.2). Such an improvement could also be used in the refinement phase, but we found experimentally that this would have no noticeable effect.

5.4 Two Improvements to PLP

In this section, we first show how to compute an upper bound of aggregate burstiness of flows and how to include some non-convex service curves in the PLP analysis. Note that our two improvements

concern $\text{PLP}^{\text{backlog}}_{f,i}$ and $\text{PLP}^{\text{delay}}_{f,i}$ in step 3) of PLP as explained in Section 5.2.3; specifically, in this section, we assume that we have a collection of trees where TFA per-node delay bounds and bounds on the burstiness of flows at cuts are already obtained.

Let us first briefly present the linear programs used by PLP, more details can be found in Appendix 5.B. PLP considers the arrival and departure time of a bit of interest; it then derives a number of time instants at every node, each of which is represented by a variable in PLP. To every time instant at a node is also associated with a variable that represents the value of the cumulative arrival function of the flows at this time instant. Network calculus relations such as arrival curve constraints, FIFO constraints, and service curve constraints are translated into linear constraints; the objective function to be maximized is the delay or backlog of the flow of interest.

Recall that at this step, we use the cut network and thus assume that the graph induced by flows is a collection of non-connected trees, and the analysis is done on every tree (with edges directed towards the root). It follows that each node v, except the root, has a unique successor. We let $\text{sc}(v)$ denote the successor of node v, and add an artificial node v_0 to be the successor of the root. Define the depth of nodes dp as follows: $\text{dp}(v_0) = 0$ and for every v, $\text{dp}(v) = \text{dp}(\text{sc}(v)) + 1$.

Time Variables: For every node v, define $\mathbf{t}_{(v,k)}$ with $k \in \{0, \dots, \text{dp}(v)\}$.

Process Variables: For every node v and v' and every f at v, define $\mathbf{Ft}^{v,f}_{v',k}$ with $k \in \{0, \dots, \text{dp}(v')\}$, where $\mathbf{Ft}^{v,f}_{v',k}$ is a variable for the cumulative arrival function of flow f at the input of node v at time $\mathbf{t}_{(v',k)}$. In the next paragraphs, we only present the parts of the linear program that are modified. The complete linear programs are presented in Appendix 5.B.

5.4.1 PLP to Upper-bound the Aggregate Burstiness of Flows

We show that the same PLP, used to compute a backlog bound for a single flow, can be used to compute a backlog bound for the aggregate with some modifications: Consider a set of flows of interest F, whose destination is the root of the tree. Modify the PLP used to compute a backlog bound for a single flow as follows: Let v_f be the first node visited by flow f in the tree for all $f \in F$.

- Additional constraints: $\forall f \in F, \forall k,\ k \in [0,\ \text{dp}(v_f)],\ \mathbf{Ft}^{v_f,f}_{v_0,0} - \mathbf{Ft}^{v_f,f}_{v_f,k} \leq b_f + r_f(\mathbf{t}_{(v_0,0)} - \mathbf{t}_{(v_f,k)})$;
- New objective: Maximize $\sum_{f \in F} \left(\mathbf{Ft}^{v_f,f}_{v_0,0} - \mathbf{Ft}^{v_0,f}_{v_0,0} \right)$.

We call $\text{PLP}^{\text{backlog}}_{v,c}$ the resulting program when applied to class c and to the sub-tree of the cut network rooted at some node v. Here F is the set of flows that exit node v, and the program obtains the aggregate burstiness of flows of class c at the output of node v; the results are used by $\text{DRRservice}^{\text{outputBurst}}_{v}$ in the refinement phase to compute DRR strict service curves. We also apply this program when e is an edge, and also call it $\text{PLP}^{\text{backlog}}_{e,c}$. Here, F is the set of flows that use edge e and the sub-tree is rooted at the node that edge e exits. The results are used in the refinement phase by $\text{perNodeDelay}_{v,c}$ to compute per-node delay bounds.

Theorem 5.1 (PLP to Upper-bound the Aggregate Burstiness of Flows)**.** *The solution of $\text{PLP}^{backlog}_{g,c}$ is a valid bound on aggregate burstiness of flows carried by edge or node g.*

The proof is in Section 5.7.1, and the key idea of the proof is as follows: We show that a valid backlog bound for an aggregate of some flows is also a valid burstiness bound for the aggregate. This is obtained

in Lemma 5.1 where we show that in any acceptable trajectory scenario of the system (i.e., a set of valid input/output processes for all flows), the burstiness of the aggregate never exceeds the backlog bound.

5.4.2 iPLP: a PLP that Supports Non-Convex Service Curves

Our goal here is to modify PLP, used to compute a delay, such that it can handle a non-convex service curve expressed, at node v and class c, as $\max(\beta_c^v, \beta^{\text{nc},v})$, where β_c^v is piece-wise linear convex (i.e., $\beta_c^v = \max_p \beta_{R_p^v, T_p^v}$)), and $\beta_c^{\text{nc},v} = \min(\beta_{R^v,T_1}, q_c^v)$ as described in Section 5.2.1. Similar to the previous case, we present only the parts of the linear program that are modified, namely the service constraints.

For the sake of concision, we now introduce the variables and constraints $\mathbf{At}_u^v = \sum_{f \in \text{In}(v)} \mathbf{Ft}_{u,\text{dp}(u)}^{u,f}$ and $\mathbf{At}_v^v = \sum_{f \in \text{In}(v)} \mathbf{Ft}_{v,\text{dp}(v)}^{v,f}$, where $u = \text{sc}(v)$.

The original service curve constraints of PLP are kept:

$$\mathbf{At}_u^v - \mathbf{At}_v^v \geq 0; \tag{5.5}$$

$$\forall p,\ \mathbf{At}_u^v - \mathbf{At}_v^v \geq R_p^v \left(\mathbf{t}_{(u,\text{dp}(u))} - \mathbf{t}_{(v,\text{dp}(v))} - T_p^v \right). \tag{5.6}$$

As $\beta^{\text{nc},v}$ is the minimum of a rate-latency function and a constant (see Fig. 5.3); the implementation of $\beta^{\text{nc},v}$ requires a "if then else" structure: If $\mathbf{t}_{(u,\text{dp}(u))} - \mathbf{t}_{(v,\text{dp}(v))} \leq T_2$, then we must have $\mathbf{At}_u^v - \mathbf{At}_v^v \geq R^v(\mathbf{t}_{(u,\text{dp}(u))} - \mathbf{t}_{(v,\text{dp}(v))} - T_1)$. Else, if $\mathbf{t}_{(u,\text{dp}(u))} - \mathbf{t}_{(v,\text{dp}(v))} > T_2$, we must have $\mathbf{At}_u^v - \mathbf{At}_v^v \geq q_c^v$. This can be modeled by means of a binary variable in a linear program. Define $\mathbf{b}_v \in \{0,1\}$ and consider a large enough positive M, and add the following constraints:

$$\mathbf{At}_u^v - \mathbf{At}_v^v \geq R^v(\mathbf{t}_{(u,\text{dp}(u))} - \mathbf{t}_{(v,\text{dp}(v))} - T_1) - M\mathbf{b}_v; \tag{5.7}$$

$$\mathbf{At}_u^v - \mathbf{At}_v^v \leq q_c^v + M\mathbf{b}_v; \tag{5.8}$$

$$\mathbf{At}_u^v - \mathbf{At}_v^v \geq q_c^v - M(1 - \mathbf{b}_v). \tag{5.9}$$

We let $\text{iPLP}_{f,c}^{\text{delay}}$ denote this Integer, Polynomial-sized Linear Program. Theorem 5.2 is proved in Section 5.7.2.

Theorem 5.2 (iPLP: a PLP that supports non-convex service curves)**.** *Consider iPLP as constructed above. Then, 1) iPLP gives a valid delay bound for the flow of interest, and 2) the bound is less than or equal to that of PLP.*

Note that iPLP solves a Mixed-Integer Linear Programming (MILP), and the MILP solver we use does not guarantee that it finds the optimal solution. However, it guarantees that the solution is feasible, and it indicates whether the obtained solution is optimal. In all examples we tested, we always obtained the optimal solution (see Fig. 5.10 (b)).

5.5 Our Proposed Method: PLP-DRR

In this section, we provide the details of our generic method and we present two concrete implementations.

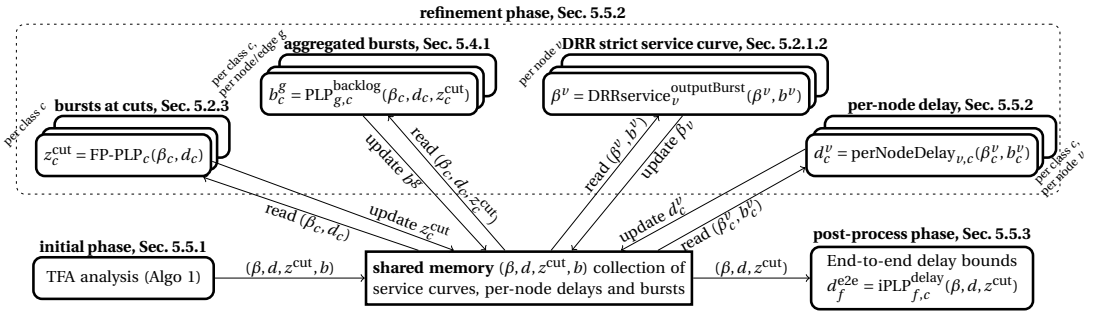


Figure 5.5: Overview of the method. The refinement phase consists in applying, in any order, any of the four types of refinement blocks shown at the top of the figure, which each improves the bounds stored in the shared memory. The refinement phase may be stopped using any criterion, such as convergence of the variables in the shared memory or a timeout. If stopped at convergence, the value of the shared memory is always the same, regardless of the order of the refinements.

5.5.1 Initial Phase: TFA-DRR

The goal of the first phase is to provide valid values for $(\beta, d, z^{\text{cut}}, b)$, using TFA. Specifically, we want to analyze the original (uncut) network using TFA (note that TFA itself does not necessarily require cuts [64]). In networks with cyclic dependencies, the TFA analysis of a DRR system of [108] cannot be directly applied here, as propagated burstiness bounds are needed to compute the DRR strict service curves and vice-versa. However, it is possible to first compute DRR strict service curves without assumption of the arrival curves using $\beta_c^{\text{CDM},v}$ for each node v and class c: these service curves only depend on the fixed parameters such as assigned quanta, the aggregate strict service curve, and maximum packet sizes of classes at a node, as explained in Section 5.2.1.1. From there, one can iterate between the computation of output bursts (used to deduce the arrival curves) and the DRR strict service curves that take into account the arrival curves.

The method is described in Algorithm 5.1: The local variable z represents the propagated burstiness of all flows at all outputs. First, at line 1, initial strict service curves of DRR in degraded operational mode are computed at all nodes for all classes. Then, the algorithm alternates between performing a TFA analysis and computing new DRR strict service curves. More precisely, at line 3, a TFA analysis (explained in Section 5.2.2) is performed for each class, with the previously computed service curves; hence, some bounds on propagated burstiness of flows z and per-node delay bounds d are computed and used to compute arrival curves at each node for each class (line 5). Then, at line 6, DRR strict service curves are improved by taking into account these arrival curves. This procedure continues until we reach stopping criteria; for example, when each component of vector d decreases insignificantly. Note that computed bounds are valid at each iteration. At this point, TFA analysis is completed, however, we need to compute bounds on the aggregate burstiness of flows of every class either at each edge (including at the cutset) and at the output of every node, as this is used in the refinement phase. This is performed at lines 7-11.

The delay and burstiness bounds computed by the TFA analysis at line 3 might not be finite. For example, after the first execution of line 3, some classes might provide finite delay bounds (called stable classes) and other infinite delay bounds (called unstable classes). At the first execution of lines 4-6, the DRR strict service curves of the unstable classes are improved using the arrival curves of the stable classes. Then, at the next iteration, more stable classes might be obtained, and so on.

Algorithm 5.1: Initial Phase: TFA-DRR

Result: Initial values of $(\beta, d, z^{\text{cut}}, b)$
Local Variable: z, collection of bounds on the propagated burstiness of flows

1 **for** *node* $v \leftarrow 1$ *to* $|\mathcal{V}|$ **do** $\beta^v \leftarrow \beta^{\text{CDM},v}$;
2 **while** *Stopping criteria not reached* **do**
3 **for** *each class* c **do** $(d_c, z_c) \leftarrow \text{GenericTFA}_c\left(\beta_c\right)$;
4 **for** *node* $v \leftarrow 1$ *to* $|\mathcal{V}|$ **do**
5 **for** *each class* c **do** compute α_c^v from $z_c^{\text{In}_c(v)}$;
6 $\beta^v \leftarrow \text{DRRservice}_v^{\text{inputArrival}}(\alpha^v)$;
7 **for** *each class* c **do** $z_c^{\text{cut}} \leftarrow z_c^{E_c^{\text{cut}}}$;
8 **for** *node* $v \leftarrow 1$ *to* $|\mathcal{V}|$ **do**
9 **for** *each class* c *and each* $e \in In_c(v)$ **do**
10 $b_c^e \leftarrow \sum z_c^e$;
11 $b_c^v \leftarrow \sum z_c^{\text{Out}_c(v)}$;
12 **return** $(\beta, d, z^{\text{cut}}, b)$

Theorem 5.3 (Correctness and Convergence of TFA-DRR)**.** *Consider a network with DRR scheduling per class, as described in Section 5.1, and consider Algorithm 5.1. Then, 1)* (β, d, z) *(and the resulting* z^{cut} *and b obtained from z at lines 7-11) are valid bounds at every iteration at lines 2-6, and 2) they converge as the number of iterations goes to infinity. Note that some values of* d, z *(and the resulting* z^{cut} *and b) might be infinite.*

The proof is in Section 5.7.3. At this point we have obtained a value of $(\beta, d, z^{\text{cut}}, b)$ that constitute valid bounds.

5.5.2 Refinement Phase: PLP and Parallelization

The next phase of the method is to improve the value of $(d, \beta, z^{\text{cut}}, b)$ using the PLP methodology. As mentioned in Section 5.3, the variables $(\beta, d, z^{\text{cut}}, b)$ are interdependent, and can be improved by some refinements that we list here:

- $z_c^{\text{cut}} \leftarrow \text{FP-PLP}_c(\beta_c, d_c)$: computes burstiness bounds at cuts for class c (Section 5.2.3);
- $b_c^g \leftarrow \text{PLP}_{g,c}^{\text{backlog}}(\beta_c, d_c, z_c^{\text{cut}})$: computes burstiness bounds of the aggregate flows of class c at the output of node g, if $g \in \mathcal{V}$, or carried by edge g, if $g \in \mathcal{E}_c$ (Section 5.4.1);
- $\beta^v \leftarrow \text{DRRservice}_v^{\text{outputBurst}}\left(\beta^v, b^v\right)$: computes the DRR strict service curves at node v given the output burstiness bounds at this node (Section 5.2.1).
- $d_c^v \leftarrow \text{perNodeDelay}_{v,c}(\beta_c^v, b_c)$: computes the per-node delay of node v for class c. This is the horizontal distance between the arrival curve and the service curve. For each input edge e of node v, in addition to the aggregate burstiness b_c^e, a rate limitation imposed by the link can be used to improve the arrival curve. This is known as line-shaping [42, 63, 68]. Then, we take into account the effect of line shaping and of the packetizer as in Section VI-B of [119].

All these refinements can be applied in any order, and it is not clear in what order we should use. We avoid the issue by first presenting a generic scheme that uses a distributed computing model with a shared memory, and we show that the values converge to the same values regardless of the order of the operations under mild assumptions. We also then present two practical, concrete implementations of this scheme with parallel-for loops.

5.5.2.1 Generic Scheme: A Distributed Computing Model with Shared Memory

The generic scheme is presented in Fig. 5.5. We consider a distributed system with a shared memory and a finite number of workers (processes or threads). The shared memory stores the current value of $(\beta, d, z^{\text{cut}}, b)$; it is initialized with the result of the initial phase described in Section 5.5.1. Every worker has read and write access to the shared memory, and whenever a worker is free and decides to work, it performs the following steps:

- It chooses a refinement in the list above, let us call it h; for example, it may choose FP-PLP$_c()$ for some class c, or $\text{PLP}^{\text{backlog}}_{g,c}()$ for some c, g, etc.
- The worker then makes a read-only operation on the shared memory in order to obtain the value, say x, of its argument. For example, if h is FP-PLP$_c()$, then the worker reads $x = (\beta_c, d_c)$. We assume that read-only operations are atomic, i.e., the values read by the worker cannot be modified by other workers during the read operation (such that the worker has a valid snapshot).
- The worker computes $y = h(x)$, i.e., a new value of some of the bounds in the shared memory. For example, if h is FP-PLP$_c()$, the worker computes $y = z^{\text{cut}}_c$.
- The worker asks for a read/write lock on the shared memory. Such a lock prevents other workers from writing into the memory until the lock is released by this worker. Once the lock is obtained, the worker reads the current value y' of the same variables it wants to update from the shared memory, computes the component-wise minimum of y and y', writes the resulting values into the shared memory, and releases the lock. The minimum is computed because some other worker might have improved the same value during the computing time of this worker.
- The worker is now free and might decide to work again.

Note that some delay bounds and burstiness bounds might be infinite. For example, initial bounds obtained by TFA might be infinite for a class (but they might become finite after the operations of some workers).

This generic scheme does not prescribe any specific arrangement of how the workers are scheduled. But, for convergence, we assume:

(H) In a hypothetical execution of infinite duration, for every time $t > 0$ and every refinement h, there exists a time $s > t$ at which one worker starts working and chooses h.

Theorem 5.4 (Correctness and Convergence of PLP Refinement Phase of Section 5.5.2.1)**.** *Consider a network with DRR scheduling per class, as described in Section 5.1, and consider the generic method described above. Let $(\beta^t, d^t, z^{cut,t}, b^t)$ be the value of the shared memory at time $t > 0$. Then,*

1. *$(\beta^t, d^t, z^{cut,t}, b^t)$ are valid bounds; some values of (d^t, z^{cut^t}, b^t) might be infinite.*

Algorithm 5.2: $\text{DRR}_T^{\text{tree}}$: DRR Analysis of a tree

Data: T a tree component of the network, $(\beta, d, z^{\text{cut}}, b)$
Result: Updated values for $(\beta, d, z^{\text{cut}}, b)$
1 **for** *each node* $v \in T$ *in the topological order of* T **do**
2 **for** *each class* c *in parallel* **do**
3 $b_c^v \leftarrow \text{PLP}_{v,c}^{\text{backlog}}(\beta_c, d_c, z_c^{\text{cut}}, b_c)$;
4 $\beta^v \leftarrow \text{DRRservice}_v^{\text{outputBurst}}(\beta^v, b^v)$;
5 **for** *each class* c *in parallel* **do**
6 **for** $e \in \text{In}_c(v)$ *in parallel* **do**
7 $b_c^e \leftarrow \text{PLP}_{e,c}^{\text{backlog}}(\beta_c, d_c, z_c^{\text{cut}}, b_c)$;
8 $d_c^v \leftarrow \text{perNodeDelay}_{v,c}(\beta_c^v, b_c)$;
9 **return** $(\beta, d, z^{\text{cut}}, b)$

2. *The limit of* $(\beta^t, d^t, z^{cut,t}, b^t)$ *as* $t \to \infty$ *exists (call it* $(\beta^*, d^*, z^{cut,*}, b^*)$*). Some components of* $d^*, z^{cut,*}$, *and* b^* *might be infinite.*
3. *Given the initial value of the shared memory, the limit* $(\beta^*, d^*, z^{cut,*}, b^*)$ *is independent of the order and the execution time of every refinement.*

Theorem 5.4 assumes that each refinement is performed infinitely often (hypothesis H). Otherwise, obtained bounds in the limit will be larger than or equal to that obtained by our scheme.

5.5.2.2 Two Implementations of PLP Refinements

We presented the generic presentation of this phase. By Theorem 5.4, any implementation results in the same final bounds. In this section, we present two concrete implementations.

Both implementations have two main blocks: computing output burstiness bounds at cuts (with FP-PLP), and locally improving the per-node delays and DRR strict service curves. The main difference between the two implementations is when to switch between these two blocks of operations.

For each class c, after removing edges E_c^{cut} in $\mathcal{G}_c$, we obtain a collection of trees. Let $\mathcal{T}$ be the collection of trees of all classes. The second block, called $\text{DRR}_T^{\text{tree}}$, is executed in parallel on each tree $T \in \mathcal{T}$: it is described in Algorithm 5.2.

The algorithm is based on the observation that in a feed-forward topology, when service curves and per-node delay bounds are computed in the topological order, there is no need for iterations. Some operations are performed in the topological order of the nodes of the tree (the operation on one node must wait for the operation on its predecessors to be finished). For each node v, there are two steps in sequence. The former, at lines 2-4, computes the DRR strict service curve for all classes. This operation requires the refinement of the output burstiness bounds b_c^v for each class, and can be computed in parallel (lines 2-3). The latter, at lines 5-8, improves the per-node delays of node v based on the newly computed service curves. Again, this operation requires the refinements of the burstiness bounds at all input edges of the node, which can be computed in parallel (lines 6-7).

The two implementations are illustrated in Fig. 5.6. The first implementation (without the dashed arrow) alternates between the two sets of blocks (FP-PLP and $\text{DRR}_T^{\text{tree}}$). Each set of blocks is started

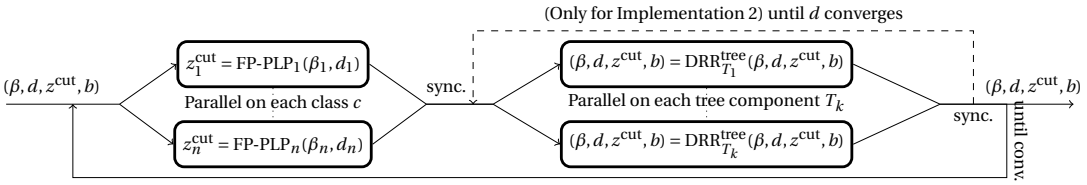


Figure 5.6: Two implementations of the refinement phases with parallelization and shared memory. Implementation 1 (without the dashed arrow): alternating between the FP-PLP and $\text{DRR}_T^{\text{tree}}$ blocks; Implementation 2 (with the dashed arrow): convergence of $\text{DRR}_T^{\text{tree}}$ before the execution of FP-PLP.

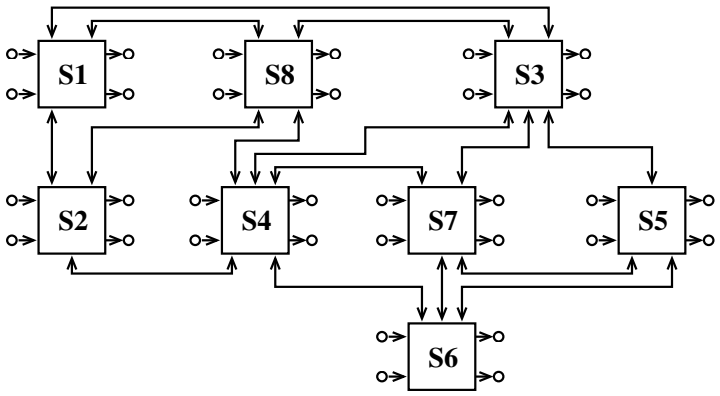


Figure 5.7: Industrial-sized network topology. The figure is taken from [31].

after synchronizing the previous set of blocks. We start by FP-PLP blocks because classes considered unstable in the initial phase (the TFA analysis outputs infinite bounds) might become stable after PLP analysis. In contrast, $\text{DRR}_T^{\text{tree}}$ cannot improve the stability region.

The second implementation tightens as much as possible the DRR service curves and per-node delays before executing again the first FP-PLP block (dashed arrow in Fig. 5.6). The aim of this implementation is to execute the FP-PLP block less often as it is more time-consuming. Indeed, it requires solving a much larger LP than in the other block. Therefore, the second block is executed several times until convergence is reached on the delay bounds.

5.5.3 Post-Process Phase: Computing the End-to-End Delay

When the refinement phase of Section 5.5.2 has converged, we proceed and compute end-to-end delay bound for each flow of interest. Strict service curves β obtained are piece-wise linear and convex. As stated in Section 5.2.1, delays can be improved when considering the non-convex strict service curve described in Fig. 5.3. Thus, we apply $\text{iPLP}^{\text{delay}}$ to compute end-to-end delay bounds.

5.6 Numerical Evaluation

We use the network of Fig. 5.7, a test configuration provided by Airbus in [42]. The industrial-sized case study of Chapter 4 is based on this network in [31]. It includes 96 end-systems, 8 switches, 984 flows, and 6412 possible paths. The rate of the links is equal to $R = 1$ Gb/s, and every switch S_i has a switching

Table 5.1: Traffic Characterization

Traffic Classes	Number of Flows	Assigned Quantum (bytes)	Maximum Packet size (bytes)
Critical	834	3070	150
Multimedia	3845	1535	500
Best Efforts	1733	1535	1535

]

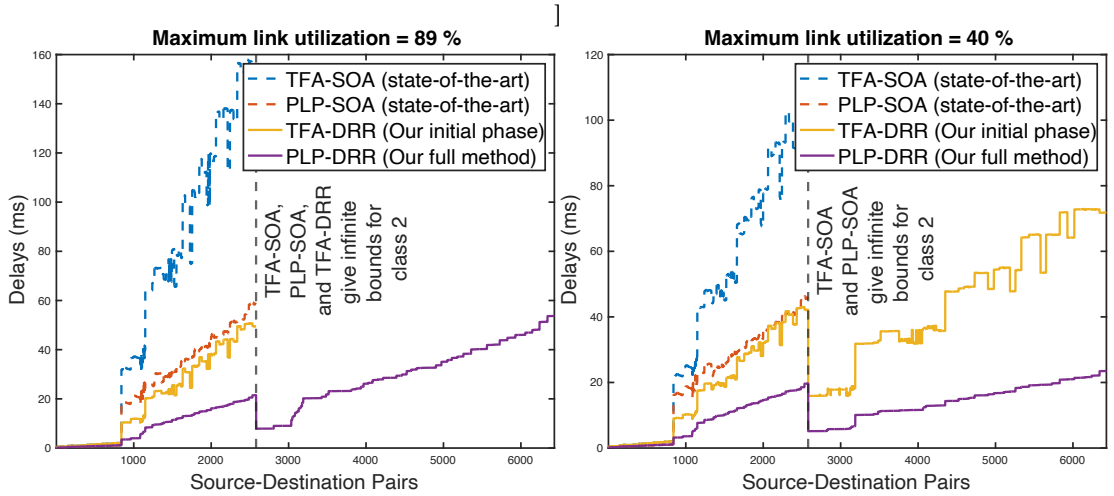
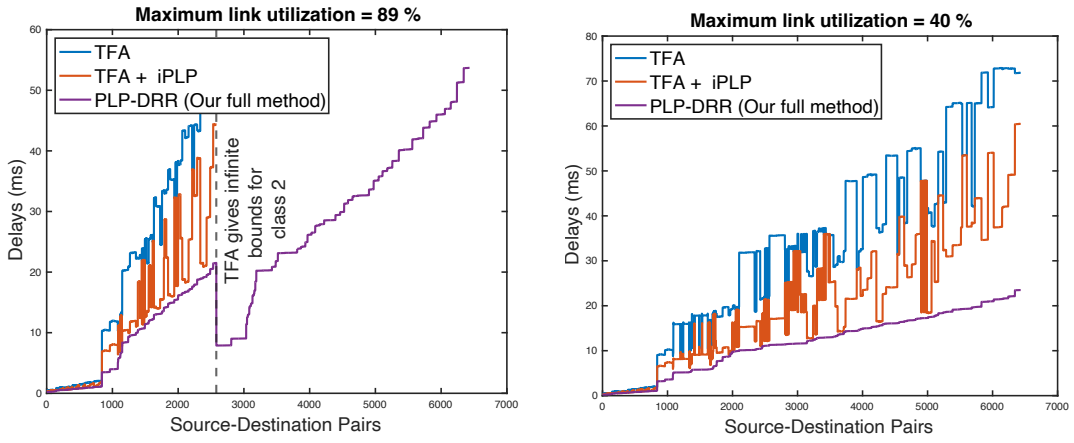
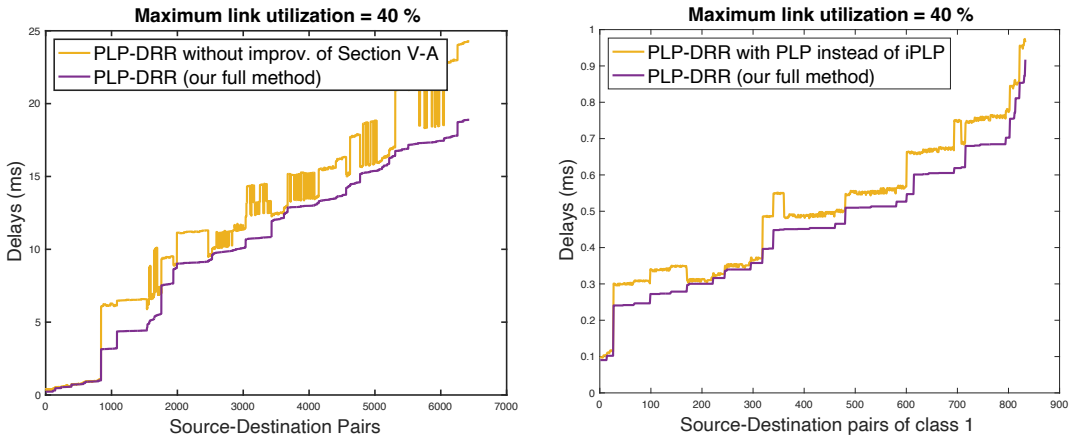


Figure 5.8: Delay bounds obtained by our methods, TFA-DRR and PLP-DRR (plain plots), and the state-of-the-art, TFA-SOA and PLP-SOA (dashed plots). TFA-SOA and PLP-SOA use DRR strict service curves in the degraded operational mode. TFA-SOA and PLP-SOA both provide infinite bounds for class 2 even when the maximum link utilization is 40%. Source-destination paths are ordered by values of our full method, and finite delays for other methods are shown first.



(a) (b)

Figure 5.9: Delay bounds of our method compared to alternative methods: comparison with 1)-2). At link utilization 89%, TFA-DRR analysis gives infinite delays for class 2. Source-destination paths are ordered by values of our full method, except in (a) where finite delays for TFA-DRR are shown first.



(a) (b)

Figure 5.10: Delay bounds of our method compared to alternative methods: comparison with 3)-4), the effect of the two PLP improvements of Section 5.4.

latency equal to 16μs. Every switch has 6 input and 6 output end-systems. There are three classes of flows: (1) critical, (2) multimedia, and (3) best-effort. Worst-case delay bounds are required for classes 1 and 2 only. There is one DRR scheduler at every switch output port with $n = 3$ classes (see Table 5.1 for details). For every flow, the path from the source to a destination can traverse at most 4 switches. In Chapter 4, as our method only applies to feed-forward networks, flow paths are chosen randomly with the constraint that graphs induced by flows are feed-forward. In this chapter, we removed this restriction and, as the network has much redundancy, this automatically generates induced flow graphs with cyclic dependencies, and is more representative of a realistic deployment. We obtain different network utilization factors by varying the minimum packet inter-arrival times. We consider two modes: when the network is lightly loaded with a maximum link utilization of 40%, and when the network is highly loaded with a maximum link utilization close to 100%.

As of today, the only methods that compute bounds on the worst-case delay of DRR networks with cyclic dependencies are TFA-SOA (see Section 5.2.2) and PLP-SOA (see Section 5.2.3) where they use DRR strict service curves in degraded operational mode (see Section 5.2.1.1). As illustrated in Fig. 5.8, our methods TFA-DRR and PLP-DRR significantly improve delay bounds and provide larger stability regions.

As our method, PLP-DRR, contains a number of improvements, we perform a numerical analysis to evaluate whether each of our improvement is useful or not. We compare our full method, PLP-DRR, to the following four alternatives:

1. TFA-DRR: we apply the initial phase only; this is the best that can be obtained with TFA and DRR service curves.
2. TFA-DRR + iPLP: we apply the initial phase and the post-process phase but do not apply the refinement phase; this shows the effect of the refinement phase.
3. Full method but we do not use our improvement in Section 5.4.1 to compute per-aggregate bounds; instead, we sum the burstiness bounds of every individual flow obtained using PLP; this shows the effect of this PLP improvement.

4. Full method but with PLP instead of iPLP in the post-process phase; this shows the effect of this PLP improvement.

In Fig. 5.9, we compare delay bounds of 1) and 2) to our full method; TFA diverges for class 2 at link utilization of 89% whereas our full method remains stable at all link utilizations below 100%. In Fig. 5.10 (a), we compare 3) to our full method. Our PLP improvements reduce the delay bounds as well as the run-times: when we use PLP per aggregate, we solve fewer PLPs. In Fig. 5.10 (b), we compare 4) to our full method. We show numerically that when delay bounds are small, iPLP captures a non-convex part of DRR strict service curves and brings an improvement compared to PLP. Experimentally, the improvements increase with the link utilization.

Note that infinite bounds might only be obtained in networks with cyclic dependencies that are highly loaded (as we assume local stability), e.g., in the industrial network we tested, when the maximum link utilization reaches 89%, TFA-DRR provides infinite bounds (see Fig. 5.9). Obtaining finite bounds implies that the network is truly stable; whereas, obtaining infinite bounds does not necessarily imply that the network is unstable, and the network might or might not be truly stable.

We use MATLAB on a 2.6 GHz 6-Core Intel Core i7 computer; thus, we have 6 workers to implement our two parallel versions of Section 5.5.2. We provide the 95% confidence interval for the run-times of the Initial phase (TFA) and two parallel implementations of the refinement phase at two different maximum link utilization; we run the program 10 times. When the maximum link utilization is 40% and 89%, 95% confidence intervals of run-times are provided in Table 5.2. Regarding the post-process phase, we compare run-times of PLP and iPLP for the flows with the longest path. We run the program 100 time and give the 95% confidence intervals in Table 5.2.

Table 5.2: Run-times

Max Link Util.	Initial Phase (TFA-DRR)	Refinement Phase Version 1	Refinement Phase Version 2	Post-Process (iPLP)	Post-Process (PLP)
40 %	[0.5,0.6] (minutes)	[5.8,5.9] (minutes)	[4.6,4.7] (minutes)	[6.4,6.5] (seconds)	[5.7,5.75] (seconds)
89 %	[3,4] (minutes)	[9,10] (minutes)	[5.5,6.5] (minutes)		

Our full method, PLP-DRR, clearly dominates the bounds and the stability region compared to our initial phase, TFA-DRR (see Fig. 5.9), but, comes with longer run times. As observed in this industrial-sized network, PLP-DRR converges relatively fast, and bounds are obtained in several minutes; also, we experimentally observe that even in a very large network (a ring-shaped topology with 20 nodes and 3 DRR classes), our method PLP-DRR converges in 6 hours; note that these run-times are obtained using a 2.6 GHz 6-Core Intel Core i7 laptop, and one should expect a fraction of these run-times in a server or a more powerful computing setup. Nevertheless, as long as PLP-DRR is feasible and has a finite run-time, say hours, the extended running time of PLP-DRR compared to TFA-DRR is not a concern. This is because, typically, finding delay bounds in a network setting is an offline problem, hence run times that are in the order of several hours are acceptable. Moreover, our bounds in the refinement phase are always valid even before convergence, hence, one can always stop iterating in the refinement phase with respect to a run-time budget and obtain delay bounds that are at least as good as those of the initial phase. Lastly, we experimentally observe that iPLP comes at a negligible cost compared to PLP (see Table 5.2) even when the network becomes very large.

5.7 Proofs

5.7.1 Proof of Theorem 5.1

We first prove Lemma 5.1: Consider a system $\mathbb{S}$ and a set of flows F that traverses $\mathbb{S}$. Let A_f (resp. D_f) denote the cumulative arrival (resp. departure) of flow $f \in F$. Let $A = \sum_{f\in F} A_f$ (resp. $D = \sum_{f\in F} D_f$) denote the arrival (resp. departure) of the aggregate of flows of interest. Assume that: (A1) System $\mathbb{S}$ is causal, i.e., $\forall (A,D) \in \mathbb{S},\ A \geq D$ and $D(t)$ only depends on $A(s)_{s\leq t}$; (A2) The departure D of system $\mathbb{S}$ is continuous; (A3) Every flow of interest $f \in F$ has a token-bucket arrival curve α_f with rate r_f and burst b_f; also, α_f is the only constraint on the arrival of the flow of interest, i.e., every cumulative arrival A_f constrained by α_f is a possible arrival for this flow; (A4) $\mathscr{B}$ is a backlogged bound for every possible arrival and departure of the aggregate of flows of interest that belongs to system $\mathbb{S}$, i.e., $\forall (A,D) \in \mathbb{S}$, we have $A - D \leq \mathscr{B}$.

Lemma 5.1. *Assume the assumptions (A1)-(A4). Then, the departure of the aggregate of the set of flows of interest F is constrained by a token-bucket arrival curve with rate r and burst $\mathscr{B}$ where $r = \sum_{f\in F} r_f$, i.e., $\gamma_{r,\mathscr{B}}$.*

Note that in a specific case where we have only one flow of interest, assumptions (A1)-(A4) are satisfied by those of Theorem 4 of [68], and both theorems give exactly the same result. Also, this result already appears in Corollary 2 of [161], unproved, and in a slightly more restrictive case.

Proof. Fix $(A_f, D_f) \in \mathbb{S}$ for every $f \in F$ and $s \leq t$. Let $r = \sum_{f\in F} r$ and $b = \sum_{f\in F} b_f$. We prove that $D(t) - D(s) \leq \gamma_{r,\mathscr{B}}(t-s) = \mathscr{B} + r(t-s)$.

We first prove that $A(s) - D(s) + H \leq \mathscr{B}$ with $H = b - \bar{b}(s)$ and $\bar{b}(s) \overset{def}{=} \sup_{u\leq s}\{A(s) - A(u) - r(s-u)\}$. Note that $\bar{b}(s)$ is the bucket size of the token-bucket $\gamma_{r,b}$ at time s, so $H \geq 0$. Define A' as follows: $A'(u) = A(u)$ for $u \leq s$ and $A'(u) = A(s) + H + r(u-s)$ for $u > s$. Observe that A' is constrained by $\gamma_{r,b}$ and $A' \geq A$. Hence, by (A3), A' is a possible arrival in $\mathbb{S}$: there exists D' such that $(A', D') \in \mathbb{S}$. Hence, by (A4), we have $A'(s^+) - D'(s^+) \leq \mathscr{B}$ where $f(x^+) = \lim_{y\to x, y>x} f(y)$. As $A'(u) = A(u)$ for $u \leq s$ and by (A1), we have $D'(u) = D(u)$ for $u \leq s$; also, by (A2), $D'(s^+) = D'(s)$. By combining this and $A'(s^+) = A(s) + H$, we obtain $A(s) - D(s) + H \leq \mathscr{B}$.

To conclude, we notice that $D(t) \overset{(A1)}{\leq} A(t) \leq A'(t) = A(s) + H + r(t-s)$, so $D(t) - D(s) \leq A(s) + H + r(t-s) - D(s) \leq \mathscr{B} + r(t-s)$. □

Theorem 5 of [68] proves the correctness of PLP. The proof consists in showing that any trajectory scenario of the network is a feasible solution of the PLP; given a trajectory scenario, the variables of the PLP (time variable and process variables) are extracted from the scenario such that all constraints are satisfied. We only add some arrival curve constraints for flows $f \in F$. In [68] it is shown that these constraints are correct for a single flow and hence the same proof can be used for each flow $f \in F$. In particular, $\mathbf{Ft}_{v_0,0}^{v_f,f}$ (resp. $\mathbf{Ft}_{v_0,0}^{v_0,f}$) represents the arrival process (departure process from the system) of flow f at time $\mathbf{t}_{(v_0,0)}$. No other constraint is modified. Only the objective function changes and maximizes the quantity of data of the flows of interests at time $\mathbf{t}_{(v_0,0)}$: $\sum_{f\in F} \mathbf{Ft}_{v_0,0}^{v_f,f} - \mathbf{Ft}_{v_0,0}^{v_0,f}$ is a backlog bound the aggregate flows and $\mathrm{PLP}_{g,c}^{\mathrm{backlog}}$ computes a backlog bound for an aggregate of flows when F is chosen to be the set of flows of class c traversing node or edge g. By Lemma 5.1, this is a bound on the output burstiness. □

5.7.2 Proof of Theorem 5.2

We proceed as explained in the last paragraph above and prove that the constraints we added are correct. Specifically, we only need to prove that a solution of the linear problem satisfies (with the notations previously introduced) $\mathbf{At}_u^v - \mathbf{At}_v^v \geq \max(\beta_c^v, \beta_c^{nc,v})(\mathbf{t}_{(u,\mathrm{dp}(u))} - \mathbf{t}_{(v,\mathrm{dp}(v))})$. From (5.5) and (5.6), $\mathbf{At}_u^v - \mathbf{At}_v^v \geq \beta_c^v(\mathbf{t}_{(u,\mathrm{dp}(u))} - \mathbf{t}_{(v,\mathrm{dp}(v))})$ holds, so we just need to focus on $\beta_c^{nc,v}$, and distinguish two cases, depending on the value of $\mathbf{b}_v$: This follows from the cases explained above (5.7) and (5.8) and the fact that M is large enough. Hence, it finishes the proof of 1). As iPLP has more constraints than PLP, the output of iPLP is less than or equal to that of PLP, which concludes 2). □

5.7.3 Proof of Theorems 5.3 and 5.4

We use the following lemma. We assume a finite set of isotone mappings $\mathscr{H}$, in $\mathscr{C} \to \mathscr{C}$ with $\mathscr{C} \subseteq (\mathbb{R}^+ \cup \{+\infty\})^I$. An *execution* of $\mathscr{H}$ is $(h_k, s_k, t_k, u_k)_{k\geq 1}$ such that: $\forall k \geq 1$, (C1) $h_k \in \mathscr{H}$; (C2) $0 < s_k \leq t_k < u_k$; (C3) $u_{k+1} > u_k$; (C4) $\forall k' \neq k$, $(t_k, u_k]$ and $(t_{k'}, u_{k'}]$ are disjoint. We also assume that (C5) Each function $h \in \mathscr{H}$ is executed infinitely many times; (C6) $\lim_{k\to\infty} s_k = \infty$. In the description of Section 5.5.2, s_k, t_k, and u_k, respectively, correspond to the reading time, locking, and unlocking times of the write operation in the execution of the k-th refinement (by order of completion times). Let $x_0 \in \mathscr{C}$ be the state of the memory at time 0. Given an execution $(h_k, s_k, t_k, u_k)_{k\geq 1}$, the state of the memory $x(t)$ evolves with time t as follows: $x(t)$ is piece-wise constant, right-continuous; it is modified at times u_k, $k \geq 1$, and $x(u_k) = \min(x(t_k), h_k(x(s_k)))$.

Lemma 5.2. *There exists x^* such that for all executions of $\mathscr{H}$ as explained above,* $\lim_{t\to\infty} x(t) = x^*$.

Proof. Let us first prove that $\lim_{t\to\infty} x(t)$ exists given $(h_k, s_k, t_k, u_k)_{k\geq 1}$. From (C2) and (C4), $u_k \leq t_{k+1} < u_{k+1}$ holds, so $x(t_{k+1}) = x(u_k)$. Then, $x(u_{k+1}) = \min(h_{k+1}(x(s_{k+1})), x(u_k)) \leq x(u_k)$, and $(x(u_k))_{k\geq 1}$ is a non-increasing sequence in $(\mathbb{R} \cup \{+\infty\})^I$, hence converges. As $\lim_{k\to\infty} u_k = +\infty$, $x(t)$ converges.

Second, we prove that the limit of x only depends on the initial value x_0. Consider two executions $(h_k, s_k, t_k, u_k)_{k\geq 1}$, and $(h'_k, s'_k, t'_k, u'_k)_{k\geq 1}$, and the state of their respective shared memory x and x', with respective limits x^* and x'^*. Set $u_0 = 0$ and define φ the following way: Set $u'_0 = 0$ and $\varphi(0) = 0$. For all $k \geq 1$, define $\varphi(k) = \min\{k' \geq 1,\ h'_{k'} = h_k \text{ and } s'_{k'} \geq u'_{\varphi(k-1)}\}$. In particular, by (C2) $u'_{\varphi(k-1)} \leq s'_{\varphi(k)} < u'_{\varphi(k)}$, so by (C3) $(\varphi(k))_{k\geq 1}$ is (strictly) increasing.

Let us now show by induction that $x(u_k) \geq x'(u'_{\varphi(k)})$ for all $k \geq 0$. The base case holds since $x(0) = x'(0) = x_0$. Let us now assume that $x(u_k) \geq x'(u'_{\varphi(k)})$.

On the one hand, $x(u_{k+1}) = \min(h_{k+1}(x(s_{k+1})), x(u_k))$. By induction hypothesis, $x(u_k) \geq x'(u'_{\varphi(k)})$. By construction $h_{k+1} = h'_{\varphi(k+1)}$ and by (C2) $s_{k+1} < u_{k+1}$, so $x(s_{k+1}) \geq x(u_k) \geq x'(u'_{\varphi(k)})$. Then, it follows that $x(u_{k+1}) \geq \min(h'_{\varphi(k+1)}(x'(u'_{\varphi(k)})), x'(u'_{\varphi(k)}))$.

On the other hand, $x'(u'_{\varphi(k+1)}) = \min(h'_{\varphi(k+1)}(x'(s'_{\varphi(k+1)})), x'(t'_{\varphi(k+1)}))$. By construction of φ, $t'_{\varphi(k+1)} \geq s'_{\varphi(k+1)} \geq u'_{\varphi(k)}$, so $x'(t'_{\varphi(k+1)}) \leq x'(u'_{\varphi(k)})$. We finally obtain $x'(u'_{\varphi(k+1)}) \leq \min(h'_{\varphi(k+1)}(x'(u'_{\varphi(k)})), x'(u'_{\varphi(k)}))$. Hence, $x(u_{k+1}) \geq x'(u'_{\varphi(k+1)})$, which proves the induction step.

Therefore, $x^* = \lim_{t\to\infty} x(t) = \lim_{k\to\infty} x(u_k) \geq \lim_{k\to\infty} x'(u'_{\varphi(k)}) = x'^*$. Inverting the roles of the two executions finishes the proof. □

Proof of Theorem 5.4. The shared memory contains non-negative numbers (burstiness bounds and

per-node delays) and piece-wise linear convex functions (the DRR strict service curves). First, observe that the piece-wise linear convex functions we deal with can be described by a finite set of elements of $\mathbb{R}^+ \cup \{+\infty\}$. As explained in Section 5.1.1, the result of DRRservice$^{\text{outputBurst}}(\beta^v, b^v)$ is the maximum of a finite number of rate-latency functions, whose rates are in a finite set, depending on fixed parameters (the arrival rates of flows) and the latencies are linearly decreasing with the output burstiness bounds b^v. Hence, the shared memory can be expressed as a family of $x = (T, d, z^{\text{cut}}, b)$.

5.7.3.1 Validity

Assuming valid initial bounds, the four types of functions used, PLP$^{\text{backlog}}$, FP-PLP, DRRservice$^{\text{outputBurst}}$ and perNodeDelay (defining set $\mathscr{H}$), provide valid bounds, proved in the literature. Moreover, if $(T, d, z^{\text{cut}}, b)$ and $(T', d', z'^{\text{cut}}, b')$ are valid bounds, then $(T \wedge T', d \wedge d', z^{\text{cut}} \wedge z'^{\text{cut}}, b \wedge b')$ are also valid bounds (where $\wedge$ is the minimum operation). This is straightforward for the per-node delays and burstiness bounds. For the service curves, it is enough to notice that the maximum of two strict service curves for a node is also a strict service curve for that node. So if $x(t_k)$ and $h_k(x(s_k))$ represent valid bounds, $\min(x(t_k), h_k(x(s_k)))$ are also valid bounds, so $x(t)$ always represents valid bounds.

5.7.3.2 Convergence

We then set $\mathscr{C}$ as the set of valid parameters for the problem and apply Lemma 5.2 where s_k, t_k, and u_k respectively correspond to the reading time, locking, and unlocking times of the write operation. (C1)–(C3) hold by definition and (C4) because of the lock operation. (C5) follows from (H). Furthermore, since there is a finite number of workers, (H) also implies that every execution completes except for at most a finite number, which implies (C6). □

Proof of Theorem 5.3. Consider lines 2-6 of Algorithm 5.1, and assume an infinite loop. The algorithm is sequential and hence is a specific case of the shared memory computing, where updates are one after the other, i.e., $\forall k, s_{k+1} > u_k$. The variables x is the collection (T, d, z), and the initial value is $+\infty$ at each coordinate except for two latencies per node and class (from line 1). Two types of functions are used: GenericTFA and DRRservice$^{\text{inputArrival}}$ (defining set $\mathscr{H}$). Note that decreasing delays and bursts decreases the latencies involved in the DRR service curves and conversely, so at each step $x(t_k) = x(s_k) \geq h_k(x(s_k))$ and $x(u_k) = h_k(x(s_k))$, which is exactly how the algorithm is updated. As a consequence, Lemma 5.2 can be applied: $x(t)$ converges in $\mathbb{R}^+ \cup \{+\infty\}$. □

5.8 Conclusion

We solved the problem of how to combine DRR strict service curves and the network analysis of PLP in order to obtain worst-case delay bounds in time-sensitive networks. Our method is guaranteed to find delay bounds that are at least as good as the state-of-the-art, and we found very significant improvements in the industrial network under study. It is based on a generic shared memory execution model, implementations of which can differ by the scheduling of the individual operations in the refinement phase. We proved that all implementations produce the same bounds. We proposed two concrete implementations and found that the latter performs faster.

5.9 Notation

Table 5.3: Notation List, Specific to Chapter 5

f, $\alpha_f = \gamma_{r_f,b_f}$	A flow, its token-bucket arrival curve
B^v	Aggregate strict service curve offered to v
E_c^{cut}	Cutset: removing E_c^{cut} creates a tree or a forest for class c
$\mathcal{G}_c = (\mathcal{V}_c, \mathcal{E}_c)$	The graph induced by flows of class c
$\mathcal{V}$, v	The set of all output ports, an output port
$\mathcal{V}_c$	The set of all output ports of class c
$\mathcal{E}_c$, e	The set of edges of class c, an edge of class c
$\text{In}_c(v) \subset \mathcal{E}_c$	The set of edges of class c that are incidents at node v
$\text{Out}_c(v) \subset \mathcal{E}_c$	The set of edges of class c that leave node v
b_c^e	Bound on aggregate burstiness of flows of class c carried by edge e
b_c^v	Bound on aggregate burstiness of flows of class c that exits node v
z_c^e	Collection of burstiness upper bounds for transit flows of class c carried by edge e
z_c^E	Collection of z_c^e such that $e \in E$
z	Collection of $z_c^{\mathcal{E}_c}$ of all classes c
b	Collection of b_c^v and b_c^e for every class c, every edge e, and every node v
d_c	Collection of delay bounds at all nodes for class c
d	Collection of d_c (per-node, per-class delay bounds)
β_c	Collection of per-node strict service curve offered to class c at all nodes
β	Collection of β_c (per-node, per-class strict service curves)
z^{cut}	Collection of z_c^{cut} for all classes c
d_c^v	Delay bound on node v for class c
β_c^v	Strict service curve offered to class c at node v
z_c^{cut}	Upper bounds on the burstiness of flows of class c at cuts E_c^{cut}
$\beta_{R,T}$	$\beta_{R,T}(t) = R[t-T]^+$, rate-latency function
$\beta_c^{\text{CDM},v}$	DRR strict service curve, no assumption on arrival curves
$\beta_c^{\text{nc},v}$	Non-convex DRR strict service curve
$\mathbb{R}^+$	Set of non-negative real numbers
$\mathscr{F}$	Set of wide-sense increasing functions $f : \mathbb{R}^+ \mapsto \mathbb{R}^+ \cup \{+\infty\}$
$\gamma_{r,b}$	Token-bucket function with $\gamma_{r,b}(0) = 0$ and $\gamma_{r,b}(t) = r\,t + b$ for $t > 0$

Appendix

5.A Detailed Background on DRR Strict Service Curves

Here we present more background on DRR strict service curves of Chapter 4, using the notations and assumptions used in this chapter, that enables a reader to implement what we use in the chapter.

5.A.1 Degraded Operational Mode

Here we present Corollary 4.2 of Chapter 4 that presents a convex strict service curve for DRR, in degraded operational mode:

Let v be a node that, shared by n classes, uses DRR, as explained in Section 5.1.1, with quantum Q_c^v for class c. The node offers a strict service curve B^v to the aggregate of the n classes. For any class c, $d_c^{\max}$ is the maximum residual deficit defined by $d_c^{\max} = l_c^{\max} - \epsilon$ where $l_c^{\max}$ is an upper bound on the packet size of flows of class c at node v and ϵ is the smallest unit of information seen by the scheduler (e.g., one bit, one byte, one 32-bit word, ...).

Then, for every c, v offers to class c a strict service curve $\beta_c^{\mathrm{CDM},v}$ given by $\beta_c^{\mathrm{CDM},v}(t) = \gamma_i^{\mathrm{convex}}(B^v(t))$ with

$$\gamma_i^{\mathrm{convex}} = \max\left(\beta_{R_c^{\max},T_c^{\max}}, \beta_{R_c^{\min},T_c^{\min}}\right) \tag{5.10}$$

$$R_c^{\max} = \frac{Q_c^v}{Q_{\mathrm{tot}}^v},\ T_c^{\max} = \sum_{c',c'\neq c}\left(Q_{c'}^v + d_{c'}^{\max} + \frac{Q_{c'}^v}{Q_c^v} d_c^{\max}\right) \tag{5.11}$$

$$R_c^{\min} = \frac{Q_c^v - d_c^{\max}}{Q_{\mathrm{tot}}^v - d_c^{\max}},\ T_c^{\min} = \sum_{c',c'\neq c}\left(Q_{c'}^v + d_{c'}^{\max}\right) \tag{5.12}$$

and $Q_{\mathrm{tot}}^v = \sum_c Q_c^v$. In (5.10), $\beta_{R,T}$ is a rate-latency function defined in Table 5.3.

For the non-convex strict service curve, $\beta_c^{\mathrm{nc},v}$, we have

$$q_c^v = Q_c^v - d_c^{\max} \tag{5.13}$$

$$T_1 = T_c^{\min} \tag{5.14}$$

5.A.2 Non-Degraded Operational Mode

Here we present a new formulation of Corollary 4.4. We slightly generalize Corollary 4.4, using Lemma 4.1, such that it enables us to take into account any available output arrival curves. Specifically, Corollary 4.4 is an application of this new formulation where we replace $\alpha^*_{c'}$ with $\alpha_{c'} \oslash \beta^{\text{old}}_{c'}$, which is an output arrival curve for class c', in (5.15). Note that $\oslash$ is the min-plus deconvolution defined in (2.9).

Let v be a node with the assumptions in Section 5.A.1. Also, assume that every class c has an output arrival curve α^*_c and a strict service curve β^{old}_c, and let $N_c = \{c_1, c_2, \ldots, c_n\} \setminus \{c\}$, and for any $J \subseteq N_c$, let $\bar{J} = N_c \setminus J$. Then, for every class c, a new strict service curve β^{new}_c is given by

$$\beta^{\text{new}}_c = \max\left(\beta^{\text{old}}_c, \max_{J \subseteq N_c} \gamma^{\text{convex}\,J}_i \circ \left[B^v - \sum_{c' \in \bar{J}} \alpha^*_{c'}\right]^+_{\uparrow}\right) \tag{5.15}$$

with

$$\gamma^{\text{convex}\,J}_i = \max\left(\beta_{R^{\max J}_c, T^{\max J}_c}, \beta_{R^{\min J}_c, T^{\min J}_c}\right) \tag{5.16}$$

$$R^{\max J}_c = \frac{Q^v_c}{Q^{J,c}_{\text{tot}}},\ T^{\max J}_c = \sum_{c' \in J}\left(Q^v_{c'} + d^{\max}_{c'} + \frac{Q^v_{c'}}{Q^v_c} d^{\max}_c\right) \tag{5.17}$$

$$R^{\min J}_c = \frac{Q^v_c - d^{\max}_c}{Q^{J,c}_{\text{tot}} - d^{\max}_c},\ T^{\min J}_c = \sum_{c' \in J}\left(Q^v_{c'} + d^{\max}_{c'}\right) \tag{5.18}$$

$$Q^{J,c}_{\text{tot}} = Q_c + \sum_{c' \in J} Q_{c'} \tag{5.19}$$

In (5.15), $[.]^+_{\uparrow}$ is the non-decreasing and non-negative closure: The non-decreasing and non-negative closure $[y]^+_{\uparrow}$ of a function $y : \mathbb{R}^+ \to \mathbb{R}^+ \cup \{+\infty\}$ is the smallest non-negative, non-decreasing function that upper bounds y. Also, $\circ$ is the composition of functions. In (5.16), $\beta_{R,T}$ is a rate-latency function defined in Table 5.3; note that a rate-latency function, as defined in Table 5.3, has a rate expressed in bit/s, and a latency expressed in seconds, however, rate and latencies defined in (5.17) and (5.18) are respectively unitless and in bits. This is because $\beta_{R^{\max J}_c, T^{\max J}_c}$ and $\beta_{R^{\min J}_c, T^{\min J}_c}$ are later composed by a function expressed in bit/s, in (5.15), hence the final results also are in bit/s and seconds.

The essence of (5.15) is as follows. Equation (5.15) gives new strict service curves β^{new}_c for every flow c; they are derived from already available strict service curves β^{old}_c and from output arrival curves of classes α^*_c; this enables us to improve any collection of strict service curves that are already obtained.

Let $\Pi^{\text{convex}}_v : \mathscr{F}^{2n} \to \mathscr{F}^n$ be the mapping at server v that maps $\left(\beta^{\text{old}}_1, \beta^{\text{old}}_2, \ldots, \beta^{\text{old}}_n\right)$ using $\left(\alpha^*_1, \alpha^*_2, \ldots, \alpha^*_n\right)$ to $\left(\beta^{\text{new}}_1, \beta^{\text{new}}_2, \ldots, \beta^{\text{new}}_n\right)$ as in (5.15). Then, an iterative scheme can be defined as in Algorithm 5.3.

In Chapter 4, we showed that for every class c , β^{old}_c and β^{new}_c are strict service curves for class c and $\beta^{\text{old}}_c \leq \beta^{\text{new}}_c$, i.e., an increasing sequence of strict service curves is obtained for every class. Also, this sequence is a guaranteed simple convergence, starting from any valid initial strict service curves. Note that the computed strict service curves at each iteration are valid and hence can be used to derive valid delay bounds; this means the iterative scheme can be stopped at any iteration. For example, the iterative scheme can be stopped when the delay bounds of all classes decrease insignificantly. The scheme requires being initialized by strict service curves. We use $\beta^{\text{CDM},v}_c$, obtained in Section 5.A.1 for the initial strict service curves at lines 1-2.

Algorithm 5.3: DRRservice$_v^{\text{inputArrival}}(\alpha_1,\ldots,\alpha_n)$

Result: Collection of strict service curves $\left(\beta_1^v,\ldots,\beta_n^v\right)$
Local Variables: Collections of strict service curves $\left(\beta_1^{\text{old}},...,\beta_n^{\text{old}}\right)$ and $\left(\beta_1^{\text{new}},...,\beta_n^{\text{new}}\right)$

```
for c ← 1 to n do
    β_c^old ← β_c^{CDM,v} ;
while Stopping criteria not reached do
    for c ← 1 to n do
        α_c^* ← α_c ⊘ β_c^old ;
    β^new ← Π_v^convex(α^*, β^old);
β^v ← β^new;
return (β_1^v, ..., β_n^v)
```

When every class c has a token-bucket arrival curve at the output, say γ_{r_c,b_c^v}, and a known strict service curve β_c^v, DRRservice$_v^{\text{outputBurst}}\left(\beta^v,b^v\right)$ is the function that implements (5.15) (i.e., Π_v^{convex} defined above) and returns an improved collection of strict service curves for all classes at node v.

5.B Detailed Background on PLP

Here we present more background on PLP of [68], using our notations, which enables a reader to implement what we use in the chapter. Specifically, we summarize linear programs used by PLP. For the rest of the section, a reader is invited to recall the definitions of Section 5.4. Readers who know reference [68] can map our notations to those of [68] and vice-versa as follows: For time variables, map $\mathbf{t}_{(j,k)}$ of [68] to $\mathbf{t}_{(v,k)}$ by mapping j to v. For process variables, map $\mathbf{F}_i^j\mathbf{t}_{(j',k)}$ of [68] to $\mathbf{Ft}_{v',k}^{v,f}$ by mapping j to v, i to f, and j' to v'.

Note that linear programs of [68] contains some constraints obtained from the Single Flow Analysis (SFA) delay bounds [40]. However, in practice, such constraints have no or negligible effects as SFA bounds are often dominated by those of TFA. Hence, in this chapter, we do not use SFA constraints.

5.B.1 PLP$_{f,c}^{\text{delay}}$: A PLP That Computes an End-to-end Delay Bound for a Single Flow

The goal of PLP$_{f,c}^{\text{delay}}(\beta_c,d_c,z_c^{\text{cut}})$ is to find a valid end-to-end delay bound for a flow of interest f that belongs to a class c. We assume a sub-tree of the cut network where the root is the sink server of the flow of interest f (i.e., the last server is traversed by flow f). Recall that we add an artificial node, node v_0, that is the successor of the root. We call $\mathcal{V}_c^f$ the set of output ports in this sub-tree. We assume that the burstiness of flows at cuts is given, i.e., z_c^{cut}. Also, for each node v, a convex, piece-wise linear service curve (i.e., β_c) and a per-node delay bounds (i.e., d_c) are provided.

5.B.1.1 Constraints

In the constraints we define below, let server $u=\text{sc}(v)$ be the successor of server v. Also, we denote flows by g, not to be confused by the flow of interest f.

• Time Constraints:

- $\forall v \in \mathcal{V}_c^f,\ \forall k \in [0,\ \mathrm{dp}(v)-1], \mathbf{t}_{(v,k)} \geq \mathbf{t}_{(v,k+1)}$;
- $\forall v \in \mathcal{V}_c^f,\ \forall k \in [0,\ \mathrm{dp}(v)], \mathbf{t}_{(v,k)} \leq \mathbf{t}_{(u,k)}$.

• FIFO Constraints:

- $\forall v \in \mathcal{V}_c^f,\ \forall k \in [0,\ \mathrm{dp}(u)], \forall g \in \mathrm{In}_c(v), \mathbf{Ft}_{v,k}^{v,g} = \mathbf{Ft}_{u,k}^{u,g}$.

• Service Curve Constraints:

Recall that β_c^v is piece-wise linear convex (i.e., $\beta_c^v = \max_p \beta_{R_p^v, T_p^v}$), and also recall $\mathbf{At}_u^v = \sum_{g \in \mathrm{In}(v)} \mathbf{Ft}_{u,\mathrm{dp}(u)}^{u,g}$ and $\mathbf{At}_v^v = \sum_{g \in \mathrm{In}(v)} \mathbf{Ft}_{v,\mathrm{dp}(v)}^{v,g}$, where $u = \mathrm{sc}(v)$. Then, $\forall v \in \mathcal{V}_c^f$,

- $\mathbf{At}_u^v - \mathbf{At}_v^v \geq 0$;
- $\forall p,\ \mathbf{At}_u^v - \mathbf{At}_v^v \geq R_p^v \left(\mathbf{t}_{(u,\mathrm{dp}(u))} - \mathbf{t}_{(v,\mathrm{dp}(v))} - T_p^v \right)$.

• Per-node Delay Bound Constraints:

- $\forall v \in \mathcal{V}_c^f,\ \forall k \in [0,\ \mathrm{dp}(u)],\ \mathbf{t}_{(u,k)} - \mathbf{t}_{(v,k)} \leq d_c^v$.

• Arrival Curve Constraints:

For each flow g of class c, recall that v_g is the source server of flow g. Define $\bar{b}_g$, the burstiness bound of flow g, as follows: If flow g is a fresh flow, its arrival curve is a token-bucket arrival curve γ_{r_g, b_g}, as defined in Section 5.1, and thus $\bar{b}_g = b_g$. Else, flow g is a cut flow, its arrival curve is a token-bucket arrival curve $\gamma_{r_g, \bar{b}_g}$ where $\bar{b}_g$ is obtained from z_c^{cut}.

For every flow g,

- $\forall 0 \leq k < k' \leq \mathrm{dp}(v_g),\ \mathbf{Ft}_{v_g,k}^{v_g,g} - \mathbf{Ft}_{v_g,k'}^{v_g,g} \leq \bar{b}_g + r_g \left(\mathbf{t}_{(v_g,k)} - \mathbf{t}_{(v_g,k')} \right)$.

• Shaping Constraints:

For every $v \in \mathcal{V}_c^f$ and every edge $e = (v,u)$, let $F_{v,u}$ be the set of flows of class c, carried by the edge $e = (v,u)$. Then,

- $\forall 0 \leq k < k' \leq \mathrm{dp}(u),\ \sum_{g \in F_{v,u}} \left(\mathbf{Ft}_{u,k}^{u,g} - \mathbf{Ft}_{u,k'}^{u,g} \right) \leq l_c^{\max} + R^v \left(\mathbf{t}_{(u,k)} - \mathbf{t}_{(u,k')} \right)$.

• Monotonicity Constraints:

Recall that for each flow g of class c, v_g is the source server of flow g. Then, for every flow g,

- $\forall k \in [0, \mathrm{dp}(v_g) - 1],\ \mathbf{Ft}_{v_g,k}^{v_g,g} \geq \mathbf{Ft}_{v_g,k+1}^{v_g,g}$.

5.B.1.2 Objective

The Objective is $\max \left(\mathbf{t}_{(v_0,0)} - \mathbf{t}_{(v_f,0)} \right)$.

5.B.2 $\text{PLP}^{\text{backlog}}_{f,c}$: A PLP That Computes a Backlog Bound for a Single Flow

The goal of $\text{PLP}^{\text{backlog}}_{f,c}(\beta_c, d_c, z_c^{\text{cut}})$ is to find a valid backlog bound for a flow of interest f that belongs to a class c. By [68, Theorem 4], the objective of this linear program, is a bound on the burstiness of flow f at the output of node v.

5.B.2.1 Constraints

This linear program contains all the constraints of $\text{PLP}^{\text{delay}}_{f,c}$, defined in Section 5.B.1, with the following changes:

First, for shaping constraints, we should remove the flow of interest f from $F_{v,u}$. Specifically, we should replace $F_{v,u}$ by $F_{v,u} \setminus \{f\}$, i.e., the set of flows of class c, carried by the edge $e = (v,u)$, excluding flow of interest f.

Second, we add the following constraints: Recall that v_f is the source of flow of interest f.

- $\forall k \in [0, \text{dp}(v_f)]$, $\mathbf{Ft}^{v_f,f}_{v_0,0} - \mathbf{Ft}^{v_f,f}_{v_f,k} \leq b_f + r_f \left(\mathbf{t}_{(v_0,0)} - \mathbf{t}_{(v_f,k)}\right)$.

5.B.2.2 Objective

The Objective is $\max\left(\mathbf{Ft}^{v_f,f}_{v_0,0} - \mathbf{Ft}^{v_0,f}_{v_0,0}\right)$.

5.B.3 FP-PLP_c: A PLP That Computes Bounds on The Burstiness of Flows at Cuts

The goal of FP-$\text{PLP}_c(\beta_c, d_c)$ is to find valid bounds on the burstiness for flows of class c at cuts, i.e., to compute valid values for z_c^{cut}. We assume for each node v, a convex, piece-wise linear service curve (i.e., β_c) and a per-node delay bounds (i.e., d_c) are provided.

Let F_c^{cut} be the set of cut flows of class c. For each flow $f \in F_c^{\text{cut}}$, we define a variable $\mathbf{x}_f$ that represents the burstiness of the arrival curve of flow f at its source. FP-PLP_c is constructed as follows: For each cut flow $f \in F_c^{\text{cut}}$, a fresh set of time and process variables is defined, and all constraints of $\text{PLP}^{\text{backlog}}_{f,c}$, defined in Section 5.B.2, are added to the set of constraints of FP-PLP_c; The common variables between constraints of different cut flows are only $\mathbf{x}_f$ variables. Then, FP-PLP_c maximizes the sum of all $\mathbf{x}_f$ variables; it is shown that in [68, Theorems 7 and 8], if the solution is bounded, the values of $\mathbf{x}_f$ in the solution, are valid bounds on the burstiness of cut flows.

5.B.3.1 Constraints

We define the constraints of FP-PLP_c as follows:

• For each fresh flow $f \in F_c^{\text{cut}}$ (i.e., flow f has an arrival curve γ_{b_f,r_f}), we add $\mathbf{x}_f \leq b_f$;

• For each transit flow $f \in F_c^{\text{cut}}$ (i.e., a flow that is not a fresh flow), we first define a fresh set of time and process variables, say $\mathbf{t}$ and $\mathbf{Ft}$, and we add all constraints of $\text{PLP}^{\text{backlog}}_{f,c}$, defined in Section 5.B.2. Also, the objective of $\text{PLP}^{\text{backlog}}_{f,c}$ is added as an constraint for $\mathbf{x}_f$: $\mathbf{x}_f \leq \left(\mathbf{Ft}^{v_f,f}_{v_0,0} - \mathbf{Ft}^{v_0,f}_{v_0,0}\right)$. Note that for arrival

curve constraints of a cut flow $g \in F_c^{\text{cut}}$, the burstiness $\bar{b}_g$ is replaced by the variable $\mathbf{x}_g$.

5.B.3.2 Objective

The Objective is $\max \sum_{f \in F_c^{\text{cut}}} \mathbf{x}_f$.

Part III

Efficient and Accurate Handling of Periodic Flows in Time-Sensitive Networks

6 Total Flow Analysis For Time-Sensitive Networks with Periodic Sources

Within time-sensitive networks, a realm profound,
Total Flow Analysis, its insights unbound.
With service curves characterized, nodes unfold,
And affine curves, their stories once told.

Yet periodic flows, a challenge we find,
Affine curves, their bounds left behind.
Enter UPP, the pseudo-periodic grace,
Capturing periodicity, a flawless embrace.

Existing tools, their limitations disclosed,
Many flows and UPP, intractability imposed.
But fear not, for a solution is near,
Finite-Horizon TFA, its path is clear.

Horizons finite, a focused gaze,
Restricting curves, traversing boundless maze.
Cyclic dependencies, their complexities untold,
Finite-Horizon TFA, its power behold.

Numerical wonders, in computations we see,
Bounds improved, where linear curves used to be.
Feasibility maintained, efficiency in sight,
As the method shines, revealing its might.

So let this chapter commence, a journey profound,
Finite-Horizon TFA, its marvels abound.
In time-sensitive networks, bounds refined,
Affine curves surpassed, a new era defined.

Created with ChatGPT, free research preview (version May 24) [141]

So far, we have addressed the service curve characterizations being, in some cases, too simple or non-existent in previous works. As explained in Section 1.3.1, in addition to service curve characterizations of network nodes, obtaining end-to-end delay bounds requires knowing arrival curve constraints of flows at their sources. It is then crucial to find accurate arrival curve constraints for flows, in particular, for periodic flows that are a common and critical type of traffic in time-sensitive networks. We assume

that flows are grouped into classes, packets inside one class are processed First-In-First-Out (FIFO) and classes are isolated using schedulers. The number of bits that flows can generate is limited at sources by arrival curve constraints and the service offered by a node to a class is characterized by means of a service curve.

Total Flow Analysis (TFA) [62, 63] obtains end-to-end delay bounds in FIFO networks and can be applied to per-class networks that are FIFO per class, where a service curve is known at every node and an arrival curve is known for every flow at the source. When the network is feed-forward, validity and correctness of TFA are shown with arrival curves and service curves of generic shapes. TFA is extended to networks with cyclic dependencies by FixPoint-TFA (FP-TFA) [119] and its variants such as SyncTFA [64], however with the restriction that arrival and service curves should be linear (i.e., token-bucket arrival curves and rate-latency service curves). Such a restriction often results in bounds that are pessimistic as they cannot accurately abstract the arrival model and the service model. For instance, for periodic flows, common in real-time and time-sensitive networks, a pseudo-periodic arrival curve, a stair function, exactly captures the periodic behavior of the flow traffic (see Fig. 6.1.1). As another example, non-convex service curves for schedulers such as Deficit Round-Robin (Chapter 4), Weighted Round-Robin (Chapter 3), and Credit Base Shaper [162] are known to improve delay bounds compared to rate-latency ones, as they accurately capture the scheduler's behavior (see Fig. 6.1.1). For time-sensitive networks, as in the context of IEEE TSN and IETF DetNet, cyclic dependencies are linked to certain primary properties, such as improving availability and decreasing reconfiguration effort, hence they are important and cannot be ignored. However, as of today, methods that analyze networks with cyclic dependencies are restricted to only linear curves, hence are potentially pessimistic.

Our first step is to generalize the theory of FP-TFA to arrival and service curves of generic shapes, which provides tighter bounds for networks with cyclic dependencies. Specifically, we present a new version of FP-TFA, called Generic FP-TFA (*GFP-TFA*), and prove its validity and correctness for arrival and service curves with generic shapes (Theorem 6.3). GFP-TFA is not a practical algorithm when there are many periodic sources, as we explain next; however, it serves as a theoretical reference; furthermore, it is used as a building block in the main algorithm later presented in this chapter.

Tools such as RTaW [92], Nancy [93], DiscoDNC [94], etc. use infinite precision arithmetic (with rational numbers) and implement Ultimately Pseudo-Periodic (UPP) curves; UPP curves have a transient part at the beginning, followed by a periodic pattern. UPP curves are of interest in practice as they have a finite representation, and moreover, they capture periodic behaviors (see Fig. 6.1.2). For instance, in the case of periodic flows, UPP curves can accurately describe their periodic arrival curve, and in the case of service curves, UPP curves can describe non-convexity. However, applying GFP-TFA with many flows and UPP curves quickly becomes intractable. This is because when aggregating several UPP curves, the pseudo-period of the aggregate function might become extremely large; moreover, the required memory to store the aggregate function might explode as the number of segments required to describe the aggregate function quickly grows (see Fig. 6.2.1). An example where this issue occurs is the avionic onboard communication system analyzed in [95]. More industrial examples can be found in Section 7.5.

An attempt to overcome this issue consists in replacing periods by smaller values such that the hyper-periods remain small, e.g., when aggregating three UPP curves with pseudo-periods equal to 3, 4, and 8, the hyper-period becomes 24; but, the pseudo-period 3 can be safely replaced by 2 thus the hyper-period becomes 8. However, this increases the load, hence the bounds, and moreover, it is not robust: It should be reapplied whenever a change happens, e.g., a period changes or a new periodic flow is added. Also, if a network is highly loaded, tweaking periods might violate local stability and the

network analysis fails. This is why, for tractability, TFA is generally applied with linear arrival curves. In summary, on the one hand, we have UPP curves that provide good bounds, but applying GFP-TFA with many periodic flows and UPP curves might be practically intractable; on the other hand, we have linear curves, which are very tractable but provide less good bounds. Authors in [163] show that the network-calculus delay-computation (i.e., horizontal deviation) only depends on a finite part at the beginning of the arrival curve and the service curve and not the complete curves. This motivates us to find a middle point, namely, curves that follow original, UPP curves up to a finite horizon, and beyond that, follow simpler, linear curves. An Ultimately Affine (UA) curve can exactly capture this: a UA curve has a transient part at the beginning, followed by a linear curve (see Fig. 6.1.2); working with UA curves mitigates the description complexity of UPP curves for an aggregate curve (see Fig. 6.2.1). Moreover, as UA curves are a subset of UPP curves, one can use a UPP implementation to handle UA curves. The main problem is now how to carefully construct such UA curves, from original, UPP curves and their linear upper/lower bounds, such that the end-results are not affected, i.e., as good as those obtained with original, UPP curves.

In order to solve this problem, we propose a second new version of TFA, called *Finite Horizon Total Flow Analysis (FH-TFA)*, which can be viewed as an efficient, practical replacement for GFP-TFA, and hence can be applied to networks with cyclic dependencies. The method computes sufficient finite horizons for every UPP arrival and service curve by adopting the compact domains of [163]. Note that the results about compact domains in [163] that are presented for a single node do not directly apply to TFA or FP-TFA; indeed, in a network analysis, arrival curves of flows increase as flows go deeper into the network, hence the compact domains required for delay computations increase as well. To address this, FH-TFA first applies FP-TFA using linear curves; this is very fast and provides enough information to compute sufficient finite horizons. Next, it constructs UA curves where the duration of the transient part is set to the computed sufficient finite horizons (see Fig. 6.4.1). Last, it applies GFP-TFA with these UA curves; the complexity is small due to the replacement of UPP curves by UA curves. In the common case where service curves are super-additive, we prove that FH-TFA produces the very same bounds as the intractable GFP-TFA (Theorem 6.4). Since FH-TFA is considerably less complex, this provides a tractable, efficient solution to the analysis of networks with UPP curves.

The contributions of this chapter are as follows:

- We develop and validate FH-TFA, an algorithm that provides delay bounds for deterministic networks with generic topology, and generic arrival and service curves that are implemented as UPP or UA curves (Theorem 6.4). In contrast, for networks with cyclic dependencies, existing versions of TFA are limited to linear arrival and service curves, which may affect the quality of the delay bounds.
- We develop and validate GFP-TFA, an algorithm that generalizes the theory of existing versions of TFA (FP-TFA) to arrival curves and service curves of generic shapes, and provides tighter bounds for networks with generic shapes (Theorem 6.3). GFP-TFA is used to derive a building block of FH-TFA and to establish the validity of FH-TFA. GFP-TFA is of independent interest, but in practice applying it with many periodic sources with different periods might be intractable. In the common case where the service curves are super-additive, FH-TFA produces the very same bounds as the GFP-TFA (item (2) of Theorem 6.4) but at considerably less complexity.
- FH-TFA always provides valid delay bounds that are guaranteed to be less than or equal to those obtained by FP-TFA (item (1) of Theorem 6.4).
- FH-TFA is thus the only known method that obtains formally proven delay bounds with TFA,

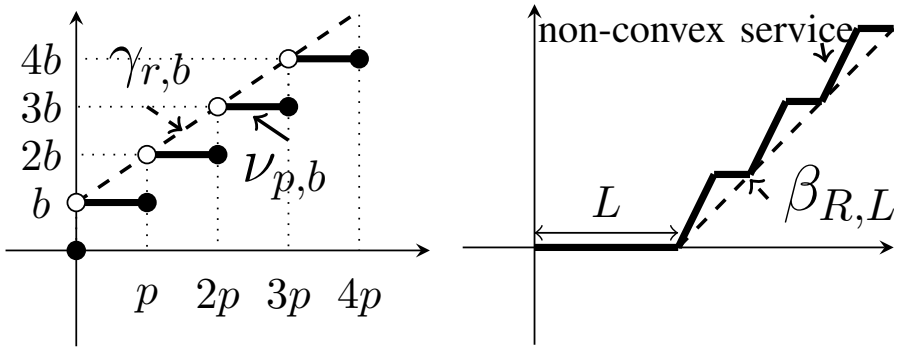


Figure 6.1.1: Left: the stair function $\nu_{b,p} \in \mathscr{F}$ defined for $t \geq 0$ by $\nu_{b,p}(t) = b\left\lceil \frac{t}{p} \right\rceil$ and token-bucket function $\gamma_{r,b} \in \mathscr{F}$ defined for $t > 0$ by $\gamma_{r,b}(t) = b + rt$ and for $t = 0$ by $\gamma_{r,b}(0) = 0$ (in the figure, we have $r = \frac{b}{p}$). Right: a non-convex service curve and a rate-latency $\beta_{R,L} \in \mathscr{F}$ that lower bounds it with $\beta_{R,L}(t) = \max(0, R(t-L))$.

and can handle many periodic sources with different periods.

We give a numerical application to real, industrial cases provided by industrial partners of RTaW, a leading company in Ethernet TSN design, performance evaluation and automated configuration tools; these examples are anonymized and slightly changed. We observe the following:

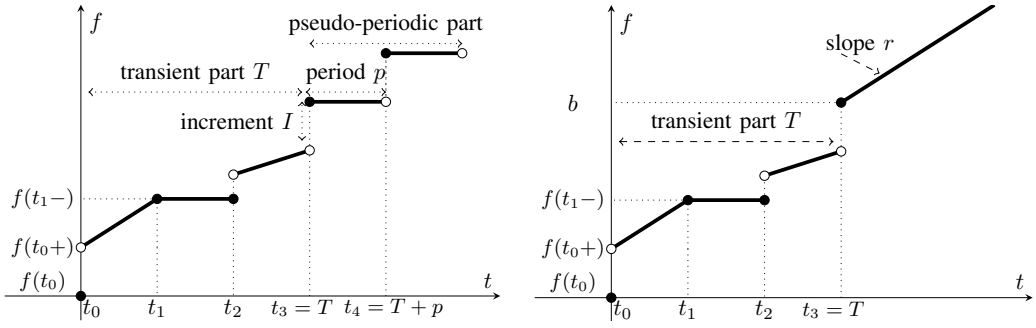
- GFP-TFA with UPP curves cannot be applied to any of the examples we tested, as it produces a memory size error.
- In contrast, FH-TFA with UPP curves applies and remains tractable in all examples we tested, even in a very large, industrial network with cyclic dependencies and many periodic flows.
- Bounds obtained with FH-TFA and UPP curves (recall that they are the same as would be obtained with GFP-TFA) are considerably less than those obtained with linear approximations of the UPP curves.

The rest of the chapter is organized as follows. In Section 6.1, we give the necessary background and state-of-the-art. In Section 6.2, we describe the problem definition, the system model, the network under study, and the resulting graph. In Section 6.3, we describe GFP-TFA, a theoretical solution to the problem at hand. In Section 6.4, we describe FH-TFA, a practical solution to the problem at hand. In Section 6.6, we present proofs of theorems. In Section 6.5, we apply FH-TFA to some industrial networks, and we give the obtained delay bounds and run-times. In Section 6.7, we conclude the chapter. A summary of notation and symbols used in this chapter are given in Section 6.8.

6.1 Background and Related Works

6.1.1 Family of Functions and Operators

In this section, we define UPP and UA functions, and explain how to represent such functions with a finite amount of information.



(a) UPP curve

(b) UA curve

Figure 6.1.2: Left: Function f, a UPP curve with a representation $\left([s_1, s_2, s_3, s_4], T, p, I\right)$ where $[s_1, s_2, s_3, s_4]$ are the 4 affine segments in $[0,\ T+p)$, T is the rank (size of the transient part), p is the pseudo-period, and I is the increment. Values outside this interval can be computed on demand by $\forall t > T,\ f(t+kp) = f(t) + kI$. Right: Function f, a UA curve with a representation $\left([s_1, s_2, s_3], T, p, I\right)$ where $[s_1, s_2, s_3]$ are the 3 affine segments in $[0,\ T)$, T is the rank, b and r are the burst and the slope of the linear part with $\forall t \geq T,\ f(t) = b + r(t-T)$.

6.1.1.1 UPP and UA Curves

We follow the terminology in [164, Definition 1]. Let $\mathscr{F}^{\text{piece-wise-linear}}$ denote the set of piece-wise linear and wide-sense increasing functions $f : \mathbb{Q}^+ \mapsto \mathbb{Q}^+ \cup \{+\infty\}$ where $\mathbb{Q}^+$ is the set of non-negative rational numbers. For $f \in \mathscr{F}^{\text{piece-wise-linear}}$:

- f is *Ultimately Affine (UA)* if there exist $T, r, b \in \mathbb{Q}^+$ such that for all $t \geq T,\ f(t) = r(t-T) + b$; T is called a rank of function f, and the smallest possible value for T is the rank of the function; r and b are called the rate and the burst of the linear part (see Fig. 6.1.2).
- f is *Ultimately Pseudo-Periodic (UPP)* if there exist $T, I \in \mathbb{Q}^+$ and $p \in \mathbb{Q}^+ \setminus \{0\}$ such that $\forall t > T$ and every non-negative integer k, $f(t+kp) = f(t) + kI$; p is called a pseudo-period and I is called an increment (see Fig. 6.1.2).

The tuple $\left(S, T, p, I\right)$ is a finite representation for a UPP function f: S represents values of f in the interval $[0, T+p)$, then values of f beyond this interval can be computed using S, pseudo-period p, and increment I. S is defined as follows: $S = [s_1, \ldots, s_k]$ is a list of affine segments where for $i \in [0, k]$, $s_i = \left(t_i, t_{i+1}, f(t_i), f(t_i+), f(t_{i+1}-)\right)$ such that $\forall t \in]t_i, t_{i+1}[$, $f(t)$ is the affine function that connects points $\left(t_i, f(t_i+)\right)$ and $\left(t_{i+1}, f(t_{i+1}-)\right)$, with $f(t+) = \lim_{\epsilon \to 0} f(t+\epsilon)$ and $f(t-) = \lim_{\epsilon \to 0} f(t-\epsilon)$. We require that (1) $t_1 = 0$, (2) there exists i_0 where $t_{i_0} = T$, and (3) $t_k < T + p$ and $t_{k+1} = T + p$. We assume that S is the minimal set, i.e., at each t_i, there is either a discontinuity or a change of slope (see Fig. 6.1.2).

The tuple (S, T, r, b) is a finite representation for a UA function f; S is defined in a similar manner as UPP functions with $t_{k+1} = T$, and $\forall t \geq T,\ f(t) = b + r(t-T)$ (see Fig. 6.1.2).

It is shown that UPP (resp. UA) functions are closed under addition, subtraction, min-plus convolution, min-plus deconvolution (see (2.5) and (2.9) for definitions), minimum and maximum of two functions [164]. Moreover, such operations are automated in tools such as RealTime-at-Work (RTaW) [92], Nancy [93], DiscoDNC [94], and etc. [164, 165, 166]; these interpreters provide efficient implementations of min-plus convolution, min-plus deconvolution, horizontal deviation, and a maximum and

minimum of functions. All computations use infinite precision arithmetic (with rational numbers), and functions are represented as UPP or UA functions.

6.1.2 FixPoint Total Flow Analysis (FP-TFA)

Total Flow Analysis (TFA) [62, 63, 64, 119] is a method for obtaining worst-case delay and backlog bounds in a FIFO network. In a network where a service curve is known at every node and an arrival curve for every flow is known at the source, one run of TFA returns a valid delay bound at every node and propagated burstiness for flows. Although TFA is simple and modular, it takes into account the effect of packetizer and line-shaping. When the graph induced by flows is feed-forward (i.e, cycle-free), each node is visited in the topological order, whereby a delay bound and output burstiness bounds of flows are computed; output burstiness of flows are used as an input by the following nodes. If the graph induced by flows has cyclic dependencies, a topological order cannot be defined; instead, an iterative method is used and a fixpoint is computed [64, 119]. The method in [119], called FP-TFA, requires first making some artificial cuts in the induced graph in order to create a feed-forward network. It then computes the estimated burstiness of flows at cuts and iterates. It is shown that, if the iteration converges, the obtained fixpoint is a valid bound on the burstiness of flows at cuts, and the network is stable. Other versions of TFA that do not make cuts are presented in [64], where it is shown that they are equivalent to FP-TFA, i.e., they provide the same bounds and stability regions [64]. For networks with cyclic dependencies, all versions of TFA assume only token-bucket arrival curves and rate-latency service curves, and the validity of the results is shown with these assumptions. In Section 6.3, we provide a new version of FP-TFA, called Generic FP-TFA (*GFP-TFA*), that can be applied with any arrival curves and service curves of generic shapes, including UPP and UA ones. When arrival curves are token-bucket and service curves are rate-latency, GFP-TFA is essentially the same as FP-TFA.

6.1.3 Compact Domains for Delay Computation

It has been observed in [95] that, for some systems, the computation of the delay bounds does not require handling the full function (i.e., all values of the function for all time t in $[0,+\infty)$), but only its values on a finite prefix domain. However, the theory in [95] lacks a proof that computation in such compact domains does not affect the accuracy of the end-results. The challenge consists in computing in each node a value h such that the computation on the compact domain $[0,h]$ is sufficient to get accurate results on this node but also on the next ones along the flow path. The intuition is the following: Consider a flow that traverses two nodes in sequence. Assume that the first node (resp. the second node) requires that the arrival curve of the flow at the input of the node is accurate in $[0,h]$ (resp. $[0,h']$). As the arrival curve of the flow increases along the path and some information is lost at propagation, the arrival curve of the flow at the input of the first node should be accurate for some $[0,h'']$ where $h'' > \max(h,h')$ is large enough.

Authors in [167] derive compact domains where they prove that the accuracy of the end-results is not affected; their results and proofs only hold for input/output relations in acyclic networks of the Greedy-Processing Component (GPC), used in Real-Time Calculus (RTC).

Authors in [163] derive such compact domains, in a more general context. They show that, at a single node where an arrival curve and a service curve are known, network calculus operations, including delay computations, can be restricted to finite domains without affecting the end-results. Here, we rewrite one of their findings that we use in the chapter, using our notation:

Theorem 6.1 (Theorem 4 of [163]). *Consider a flow constrained by an arrival curve α that traverses a node that offers a super-additive service curve β. Let α' and α'' be a lower bound and upper bound, respectively, for α, i.e., $\alpha' \leq \alpha \leq \alpha''$. Also, let β' and β'' be an upper bound and lower bound, respectively, for β, i.e., $\beta'' \leq \beta \leq \beta'$. Define*

$$h^{\alpha} \stackrel{def}{=} \max(u, v) \text{ and } h^{\beta} \stackrel{def}{=} \max\left(u + D'', v\right) \tag{6.1}$$

with

$$D' = hDev\left(\alpha', \beta'\right) \tag{6.2}$$

$$B' = vDev\left(\alpha', \beta'\right) \tag{6.3}$$

$$u = \sup_{t \geq 0}\{\alpha''(t) \geq \beta''(t + D')\} \tag{6.4}$$

$$v = \sup_{t \geq 0}\{\alpha''(t) \geq \beta''(t) + B'\} \tag{6.5}$$

$$D'' = hDev\left(\alpha'', \beta''\right) \tag{6.6}$$

Then, the horizontal deviation $hDev\left(\alpha, \beta\right)$ and the vertical deviation $vDev\left(\alpha, \beta\right)$ depend only on the values of $\alpha(t)$ for $t \in [0, h^{\alpha}]$ and $\beta(t)$ for $t \in \left[0, h^{\beta}\right]$ (see Section 2.1.1.4 for definition of super-additive service curve, see definitions 2.1 and 2.2 for hDev and vDev.)

Note that in the above, α'' and β'' are valid, safe arrival curve and service curve, respectively; however, α' and β' are unsafe arrival curve and service curve, respectively. Note that the method requires that service curves are super-additive (see Section 2.1.1.4). Authors in [163] also find such compact domains for min-plus deconvolution, and explain how to integrate such compact domains with Pay Burst Only Once (PBOO) [94] and Pay Multiplexing Only Once (PMOO) [94] principles in sink-tree networks with arbitrary multiplexing. Later, authors in [168], generalize the work of [167], using the findings of [163], to be independent of GPC's operational semantics. Lastly, authors in [40, Prop. 5.13] provide looser bounds than those of Theorem 6.1.

6.2 System Model

We consider a packet-switched network. We assume that flows are grouped into static classes, and packets inside a class are processed First-In-First-Out (FIFO). Every device represents switches or routers and consists of input ports, output ports, and a switching fabric. Each packet enters a device via an input port and is stored in a packetizer. A packetizer releases a packet only when the entire packet is received. Then, the packet goes through a switching fabric. Then, the packet, based on its class, is either queued in a FIFO-per-class queue or exits the network via a terminal port.

In the rest of the chapter, we focus on one class of interest. We assume that the service offered to the aggregate of all flows that use some output port, say n, from the exit of the packetizer (on an input port) to the transmission line fed by output port n can be modeled with a service curve β_n^{UPP}, where β_n^{UPP} is a UPP function (see Section 6.1.1). Let c_n denote the transmission rate of the line fed by output port n. Each flow f of the class of interest is constrained at source by an arrival curve $\alpha_f^{0,\text{UPP}}$, where $\alpha_f^{0,\text{UPP}}$ is a UPP function (see Figure 6.1.1). In the case of periodic flows, arrival curves are stair functions (see Fig. 6.1.1). We assume that $\alpha_f^{0,\text{UPP}}(0+) \geq l^{\max}$, where $f(t+) = \lim_{\epsilon \leftarrow 0} f(t+\epsilon)$. Also, flows are statically assigned to a path, and let path(f) denote a sequence of nodes in the path of flow f. Let flows(f)

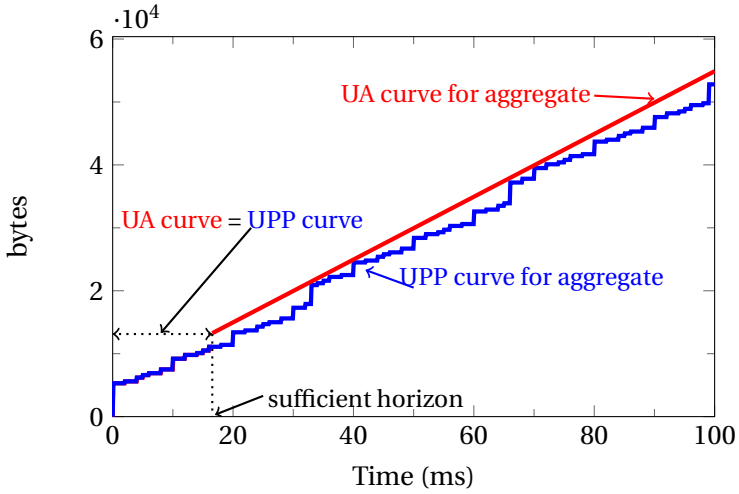


Figure 6.2.1: We consider a single node that offers a rate-latency service curve $\beta_{c,L}$ with $c = 1\text{Gb/s}$ and $L = 16\mu\text{s}$. We assume 6 fresh, periodic flows that are constrained by stair function $\nu_{p,b}$ with $p \in \{2,4,5,10,33,100\}\text{ms}$ and $b \in \{3,3,3,10,30,3\} * 100\text{bytes}$. The aggregated arrival curve has the pseudo-period equal to 3300 and moreover, 2020 segments are required to represent it; here we only plot values in $[0,100]$. Whereas, only 13 segments are required to represent the UA curve computed using our method. The sufficient horizon in this example is 16.37.

denote the set of flows at node n.

We assume that the network is locally stable, i.e., the aggregate long-term arrival rate to each output port is strictly less than the long-term service rate.

The graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ induced by flows is the directed graph defined as follows: 1) Let $\mathcal{N}$ denote the subset of all non-terminal output ports used by at least one flow. 2) The directed edge $e = (n, n') \in \mathcal{E}$ exists if there is at least one flow that traverses n and n' in this order. We say that $\mathcal{G}$ has a cyclic dependency if it contains at least one cycle. Let $E^{\text{cut}} \subseteq \mathcal{E}$ be a cut such that artificially removing the edges in E^{cut} creates a feed-forward graph. Such cuts can be obtained by any traversal graph algorithm [160]. Without loss of generality, output ports are labeled in a topological order of the cut graph, starting from output ports at the edges. Such topological orders exist as the cut network is feed-forward.

Problem Statement: The first problem is to provide, and prove the validity of, a new version of TFA that handles arrival and service curves of generic shapes in networks with generic topologies. The second problem is to apply the concept of the sufficient horizon and obtain a new version of TFA for generic topologies that remains tractable with many UPP curves.

6.3 GFP-TFA: A New Version of FP-TFA That Handles Arrival Curves and Service Curves of Generic Shapes

In this section, we present GFP-TFA, an adaptation of FP-TFA, that can be applied with arrival curves and service curves of generic shapes. GFP-TFA is an algorithm that exactly solves the problem (see Section 6.2), but, GFP-TFA has high computational complexity and in practice, applying GFP-TFA with

many UPP curves might be intractable. However, it serves as a theoretical reference, and it is used as a building block in the main algorithm, FH-TFA, presented in Section 6.4.

FP-TFA assumes that arrival curves are token-bucket, then only the burst of arrival curves increases along the path, called propagated burstiness, and the rate remains unchanged; thus, one of the variables that FP-TFA iterates on is the propagated burstiness. However, for arrival curves of generic shapes, propagated burstiness is not defined. To address this, in GFP-TFA, we replace propagated burstiness by delay-jitter bound from the source to the point of interest (variable τ): This is because an arrival curve for a flow at the point of interest is the one at the source, shifted to the left by a delay-jitter bound from the source to the point of interest; unlike propagated burstiness, this result holds for arrival curves of the generic shapes, thus GFP-TFA can be applied with any types of arrival curves. Also, FP-TFA uses a result on the effect of packetizer that is only expressed for token-bucket arrival curves, and needs to be adapted for arrival curves of generic shapes. We do this adaption in Theorem 6.2. The transformation of FP-TFA into GFP-TFA is otherwise straightforward, but for the sake of completeness, we describe GFP-TFA in details. The validity of GFP-TFA, which is less straightforward, is given in Theorem 6.3.

Recall that, as explained in Section 6.1.2, FP-TFA first cuts the network and analyzes the resulting cut network, which is feed-forward, and iterates on propagated burstiness at cuts until a fix-point is reached. We follow the same structure, with some adaptations.

6.3.1 FF-TFA: TFA for Feed-Forward Networks

FF-TFA is a building block of GFP-TFA. It applies to the feed-forward network obtained after removing a cutset and is described in Algorithm 6.1. It takes as input the collection α^0 of arrival curves of all flows at the sources, the collection β of service curves of all nodes, the cutset E^{cut}, and a collection τ^{cut} of delay-jitter bounds from source to cut for every flow that is cut. It outputs a collection d of per-node delay bounds and a collection τ of delay-jitter bounds for every flow from its source to each node in its path.

The delay-jitter bound for every flow f at every node in its path is initialized to zero (line 1); for a flow at a cut, it is initialized to the corresponding value in τ^{cut} (lines 2-4). Then, nodes are visited in the topological order of the cut network; an aggregate arrival curve at the input of node n is computed (lines 5-6) by using the function aggregateArrivalCurve$_n$ defined at line 13. This function implements the effect of line shaping (line 19) and packetizer (line 21). Line shaping [42, 156] addresses the fact that when some flows are known to arrive from the same link (i.e., carried by the same edge), a better arrival curve can be computed for the aggregate of flows; specifically, for an edge e, α_e, an aggregate arrival curve for flows carried by edge e, can be replaced by $\alpha_e \otimes \gamma_{c_e,0}$ where c_e is the maximum link speed of the edge e. For the packetizer (see Section 6.2), [119] studies the effect of packetizer when the aggregate flow is constrained by a token-bucket arrival; here we present a minor adaptation of [119, Theorem 1] that applies to arrival curves of generic shapes.

Theorem 6.2 (Output Arrival Curve at the Output of Packetizer)**.** *Consider a packetizer that is placed on a transmission line with a fixed rate c, and assume that it serves an aggregate flow, constrained by an arrival curve α; also, let l^{max} be the maximum packet size of the aggregate flow. Then, $\alpha \oslash \delta_{\frac{l^{max}}{c}}$, is an arrival curve for the aggregate flow at the output of the packetizer. Function δ_d is defined in Table 6.8.1.*

The proof is in Section 6.6.1. Note that $\alpha \oslash \delta_{\frac{l^{\max}}{c}}$ is equal to α shifted to left by $\frac{l^{\max}}{c}$, i.e., $(\alpha \oslash \delta_{\frac{l^{\max}}{c}})(t) = \alpha(t + \frac{l^{\max}}{c})$.

Combining α_n, an arrival curve for the aggregate of flows at the input of node n with β_n, the service curve offered by node n, the improved network calculus delay bound is computed as in [169] (line 8): When an output port is followed by a transmission line, authors in [169, Theorem 5] find delay bounds that improve on the classical network calculus result (which is equal to the horizontal deviation between the arrival and the service curve, i.e., $\text{hDev}(\alpha_n, \beta_n)$); this is implemented in line 8. Then, the delay-jitter of node n is added for flows at successors of node n (lines 9-11); note that a delay-jitter at node n is obtained by the subtraction of the worst-case and best-case delay bound at a node, i.e., subtraction of d_n and the transmission time of a packet of minimum size.

6.3.2 GFP-TFA

GFP-TFA is described in Algorithm 6.2: It takes as input α^0, a collection of arrival curves for each flow at the source, β, a collection of service curves offered by each node, and a cutset E^{cut} such that the cut network is feed-forward. In lines 1-7, it first computes $\bar{\tau}^{\text{cut}}$, a collection of valid delay-jitter bounds from the sources to the cuts for every flow at cuts. It initializes delay-jitter bounds from the source to the cut to zero for every flow at the cut (line 2). It then iteratively calls FF-TFA, described in Algorithm 6.1, to compute new values for delay-jitter bounds of flows from source to cut. Note that since the initial values of $\bar{\tau}^{\text{cut}}$ are 0, the scheme is monotonically non-decreasing, and thus either converges or goes to ∞. To force termination, the values are rounded up to an integer number of a chosen time resolution Δ (this is similar to rounding of burstiness to an integer number of bits in the original version of FP-TFA); then the iteration stops either when $\bar{\tau}^{\text{cut}}$ becomes stationary, or reaches a very large value called “infinite".

Then, if $\bar{\tau}^{\text{cut}}$ is finite, FF-TFA is called one last time, and hence a collection of valid, finite delay bounds at each node and a collection of valid, finite delay-jitter bounds for every flow from the source to each node in its path are obtained (because the cut network is feed-forward and locally stable). Otherwise, if the obtained $\bar{\tau}^{\text{cut}}$ bounds are infinite, GFP-TFA returns infinite bounds; in this case, the network might or might not be stable.

Theorem 6.3 (Validity of GFP-TFA)**.** *Consider a FIFO network, as described in Section 6.2, and apply Algorithm 6.2. Then,* $(\bar{d}, \bar{\tau})$ *are upper bounds on per node delay and per-flow jitter.*

The proof is in Section 6.6.2. Note that if the original network is feed-forward, the cutset E^{cut} is empty and GFP-TFA applies FF-TFA only once. When arrival curves α_f^0 of every flow f at the source are token-bucket and service curves β_n offered by every node n are rate-latency, GFP-TFA is essentially the same as FP-TFA of [119].

6.4 FH-TFA: A Practical Version of GFP-TFA

In this section, we present our second new version of TFA, FH-TFA, which provides the exact same bounds as theoretical GFP-TFA (when service curves are super-additive), while reducing the complexity.

The main difference between GFP-TFA and FH-TFA is as follows: Instead of working with original, UPP curves, FH-TFA only keeps a part of each UPP curve that affects the end-result, and beyond that, uses linear upper-bounds (resp. lower-bounds) of arrival (resp. service) curves. Specifically, FH-TFA first computes sufficient finite horizons for every curve by applying the compact domains of [163], presented in Theorem 6.1. Note that, with TFA, arrival curves of flows increase as we go deeper into the network, hence the compact domains required for delay computations increase as well. To address

Algorithm 6.1: $(d,\tau) = \text{FF-TFA}\left(\alpha^0, \beta, E^{\text{cut}}, \tau^{\text{cut}}\right)$

Input : $\left(\alpha^0, \beta, E^{\text{cut}}, \tau^{\text{cut}}\right)$, collection of arrival curves of all flows at sources, collection of service curves of all nodes, a cutset that creates a feed-forward network, and a collection of delay-jitter bounds from source to cut, for every flow that is present at a cut edge.

Output : (d,τ), a collection of per-node delay bounds and a collection of delay-jitter bounds for every flow from its source to input of every node on its path.

$\tau_{f,n} \leftarrow 0$, $\forall$flow f and $\forall$node $n \in \text{path}(n)$;
for *each edge* $e = (n', n) \in E^{cut}$ **do**
  for *each flow* f *carried by edge* e **do**
    $\tau_{f,n} \leftarrow \tau^{\text{cut}}_{f,n}$;
for *each node* n *in the topological order of the cut network* **do**
  $\alpha_n \leftarrow \text{aggregateArrivalCurve}_n\left(\alpha^0, \tau\right)$;
  $l^{\min}_n \leftarrow \min_{f \in \text{In}(n)} l^{\min}_f$;
  $d_n \leftarrow \text{hDev}\left(\alpha_n - l^{\min}_n, \beta_n\right) + \frac{l^{\min}_n}{c_n}$;
  for *each edge* $e = (n, n')$ *in the original, uncut network* **do**
    for *each flow* f *carried by edge* e **do**
      $\tau_{f,n'} \leftarrow \tau_{f,n} + (d_n - \frac{l^{\min}_n}{c_n})$;
return (d,τ);
Function $\alpha_n = \textit{aggregateArrivalCurve}_n\left(\alpha^0, \tau\right)$
  $\alpha^{\text{fresh}}_n \leftarrow \sum_{\text{fresh} f \in \text{In}(n)} \alpha^0_f$;
  for *each edge* $e \in In(n)$ **do**
    for *each flow* f *carried by edge* e **do**
      $\alpha_{f,n} \leftarrow \alpha^0_f \oslash \delta_{\tau_{f,n}}$;
    $\alpha_e \leftarrow \sum_{f \in e} \alpha_{f,n}$;
    `// Effect of line-shaping Section 6.3.1`
    $\alpha_e \leftarrow \alpha_e \otimes \gamma_{c_e,0}$;
    `// Effect of packetizer Section 6.3.1`
    $l^{\max}_e \leftarrow \max_{f \in e} l^{\max}_f$;
    $\alpha_e \leftarrow \alpha_e \oslash \delta_{\frac{l^{\max}_e}{c_e}}$;
  $\alpha^{\text{transit}}_n \leftarrow \sum_{e \in \text{In}(n)} \alpha_e$;
  $\alpha_n \leftarrow \alpha^{\text{transit}}_n + \alpha^{\text{fresh}}_n$;
  return α_n;

Algorithm 6.2: $(\bar{d},\bar{\tau}) = \text{GFP-TFA}(\alpha^0,\beta,E^{\text{cut}})$

Input : $(\alpha^0,\beta,E^{\text{cut}})$, collection of arrival curves of all flows at the source, collection of service curves of all nodes, and a cutset such that removing them creates a feed-forward network, respectively.

Output : $(\bar{d},\bar{\tau})$, a collection of valid per-node delay bound at every node and a collection of delay-jitter bound for every flow from the source to each node in its path, respectively.

1 $k \leftarrow 0$;
2 $\tau_{f,n}^{\text{cut},k} \leftarrow 0$, $\forall$flow f and $\forall e=(n',n) \in E^{\text{cut}}$;
3 **while** *($\tau^{cut,k} > \tau^{cut,k-1}$) and ($\tau^{cut,k} <$ infinite)* **do**
4 $\quad k \leftarrow k+1$;
$\quad$ // FF-TFA is described in Algorithm 6.1
5 $\quad (d,\tau) \leftarrow \text{FF-TFA}(\alpha^0,\beta,E^{\text{cut}},\tau^{\text{cut},k-1})$;
6 $\quad$ Extract new values for $\tau^{\text{cut},k}$ from τ and round them up to an integer number of the minimum resolution Δ (e.g., 1 nanosecond);
7 $\bar{\tau}^{\text{cut}} \leftarrow \tau^{\text{cut},k}$;
8 **if** *all element of $\bar{\tau}^{cut}$ are finite* **then**
9 $\quad (\bar{d},\bar{\tau}) \leftarrow \text{FF-TFA}(\alpha^0,\beta,E^{\text{cut}},\bar{\tau}^{\text{cut}})$
10 **else**
11 $\quad (\bar{d},\bar{\tau}) \leftarrow \infty$;
12 **return** $(\bar{d},\bar{\tau})$;

this, FH-TFA first applies GFP-TFA using linear curves, i.e., token-bucket arrival curves and rate-latency service curves; this is very fast and provides enough information to compute sufficient finite horizons. It then replaces every UPP curve by a UA curve that follows the UPP curve up to the computed sufficient horizon, and beyond that follows a linear upper-bound (resp. lower-bound) of the UPP arrival (resp. service) curve (see Fig. 6.4.1).

We first describe FH-TFA, and we then prove its validity and accuracy in Theorem 6.4.

6.4.1 Description of FH-TFA

FH-TFA is described in Algorithm 6.3:

- Arrival curves of all flows at the source (resp. service curves at all nodes) are lower-bounded and upper-bounded by token-bucket (resp. rate-latency) curves (lines 1-6); see Section 6.4.1.1.
- We obtain compact domains, required for delay computation, as in equation (6.1) of Theorem 6.1 at every node. As (6.1) requires to know a lower-bound and an upper-bound for the arrival curve at a node, we run two instances of GFP-TFA (using linear curves) once with safe curves, (i.e., upper-bounds of arrival curves and lower bounds of service curves), and once with unsafe curves. When this is done, we obtain compact domains at every node n, i.e., $\left(h_n^{\alpha},h_n^{\beta}\right)$ (lines 7-14). Note that if the application of GFP-TFA with safe curves returns infinite bounds (i.e., $(\bar{d}^{L''},\bar{\tau}^{L''})$ at line 7 are infinite), we then cannot find finite compact domains, and thus the rest of the algorithm is skipped, and GFP-TFA is applied with UPP, original curves (line 23). Such cases might happen in

networks with cyclic dependencies that are highly loaded.

- We then compute a sufficient horizon for the arrival curve of each flow at the source; as the arrival curve of the flow increases along its path, this horizon should be large enough such that the arrival curve of the flow respects the compact domains at every node in its path (lines 15-18); see Section 6.4.1.2.

- A UA curve is constructed, for each UPP curve, that follows the original, UPP curve up to the computed sufficient horizon, and beyond that, follows the linear upper (resp. lower for service curves) of the original, UPP curve (lines 18-20); see Section 6.4.1.3 and Fig. 6.4.1.

- Lastly, GFP-TFA is run using UA curves (line 21).

6.4.1.1 Linear Upper and Lower Bounds of UPP Curves

This section describes the four functions used in lines 2, 3, 5, and 6 of Algorithm 6.3. Note that upper-bounds and lower-bounds of the original, UPP curves can be freely chosen, as they are used in the method only to compute sufficient horizons, and they do not affect the end-results obtained by FH-TFA. We propose to use token-bucket curves (for arrival curves) and rate-latency curves (for service curves) as they can be easily computed, and are very tractable.

Consider flow f. We find two token-bucket curves $\alpha_f^{0,L'}$ and $\alpha_f^{0,L''}$ such that $\alpha_f^{0,L'} \leq \alpha_f^{0,\text{UPP}} \leq \alpha_f^{0,L''}$. Specifically, we compute the token-bucket function that upper (resp. lower) bounds $\alpha_f^{0,\text{UPP}}$, and achieves the minimum (resp. maximum) possible rate; let p and I be the period and increment of $\alpha_f^{0,\text{UPP}}$, then $\alpha_f^{0,L''} = \gamma_{r_f,b''_f}$ and $\alpha_f^{0,L'} = \gamma_{r_f,b'_f}$ with $r_f = \frac{I}{p}$ and

$$b''_f = \min_{t\geq 0}\{\alpha_f^{0,\text{UPP}} - r_f t\} \text{ and } b'_f = \max_{t\geq 0}\{\alpha_f^{0,\text{UPP}} - r_f t\} \tag{6.7}$$

Observe that the long-term rate of $\alpha_f^{0,\text{UPP}}$, which is equal to $r_f = \frac{I}{p}$, is the minimum (resp. maximum) possible rate that $\alpha_f^{0,L''}$ (resp. $\alpha_f^{0,L'}$) can achieve, otherwise they are not an upper (resp. a lower) bound of $\alpha_f^{0,\text{UPP}}$. Then, burstiness b''_f (resp. b'_f) is chosen to be as small (resp. large) as possible. In a frequent case, where $\alpha_f^{0,\text{UPP}}$ is a stair function, say ν_{p_f,b_f} (i.e., a periodic flow that sends b_f bits each p_f seconds), $r_f = \frac{b_f}{p_f}$, $b''_f = b_f$, and $b'_f = 0$ (see Fig. 6.1.1).

Consider node n. We find two rate-latency curves $\beta_n^{L''}$ and $\beta_n^{L'}$ such that $\beta_n^{L''} \leq \beta_n^{\text{UPP}} \leq \beta_n^{L'}$. Specifically, we compute the rate-latency function that lower-(resp. upper-)bounds β_n^{UPP}, and achieves the maximum (resp. minimum) possible rate; let p and I be the pseudo-period and increment of β_n^{UPP}, then $\beta_n^{L''} = \beta_{c_n,L''_n}$ and $\beta_n^{L'} = \beta_{c_n,L'_n}$ with $c_n = \frac{I}{p}$ and

$$L''_n = \min_{t\geq 0}\{t - \frac{\beta_n^{\text{UPP}}(t)}{c_n}\} \text{ and } L'_n = \max_{t\geq 0}\{t - \frac{\beta_n^{\text{UPP}}(t)}{c_n}\} \tag{6.8}$$

Observe that the long-term rate of β_n^{UPP}, which is equal to $c_n = \frac{I}{p}$, is the maximum (resp. minimum) possible rate that $\beta_n^{L''}$ (resp. $\beta_n^{L'}$) can achieve, otherwise they are not a lower (resp. an upper) bound of β_n^{UPP}. Observe that latency L''_n (resp. L'_n) is computed to be as small (resp. large) as possible.

Algorithm 6.3: $(\bar{d},\bar{\tau}) = \text{FH-TFA}\left(\alpha^{0,\text{UPP}},\beta^{\text{UPP}},E^{\text{cut}}\right)$

Input : $\left(\alpha^{0,\text{UPP}},\beta^{\text{UPP}},E^{\text{cut}}\right)$: collection of UPP arrival curves of all flows at the source, collection of UPP service curves of all nodes, and a cutset that creates a feed-forward network.

Output : $(\bar{d},\bar{\tau})$, a collection of valid per-node delay bound at every node and a collection of delay-jitter bounds for every flow from source to every node in its path.

for *each flow* f **do**
  // see Section 6.4.1.1
  $\alpha_f^{0,L''} \leftarrow \text{smallestAffineUpperBoundMinRate}(\alpha_f^{0,\text{UPP}})$;
  $\alpha_f^{0,L'} \leftarrow \text{largestAffineLowerBoundMaxRate}(\alpha_f^{0,\text{UPP}})$;
for *each node* n **do**
  // see Section 6.4.1.1
  $\beta_n^{L''} \leftarrow \text{largestRateLatencyLowerBoundMaxRate}(\beta_n^{\text{UPP}})$;
  $\beta_n^{L'} \leftarrow \text{smallestRateLatencyUpperBoundMinRate}(\beta_n^{\text{UPP}})$;
$(\bar{d}^{L''},\bar{\tau}^{L''}) \leftarrow \text{GFP-TFA}\left(\alpha^{0,L''},\beta^{L''},E^{\text{cut}}\right)$;
$(\bar{d}^{L'},\bar{\tau}^{L'}) \leftarrow \text{GFP-TFA}\left(\alpha^{0,L'},\beta^{L'},E^{\text{cut}}\right)$;
if $(\bar{d}^{L''},\bar{\tau}^{L''})$ *is finite* **then**
  for *each node* n **do**
    // see Function in Algorithm 6.1
    $\alpha_n^{L''} \leftarrow \text{aggregateArrivalCurve}_n\left(\alpha^{0,L''},\bar{\tau}^{L'''}\right)$;
    $\alpha_n^{L'} \leftarrow \text{aggregateArrivalCurve}_n\left(\alpha^{0,L'},\bar{\tau}^{L'}\right)$;
    $\left(h^\alpha,h^\beta\right) \leftarrow$ apply (6.1) in Theorem 6.1 with $\alpha' = \alpha_n^{L'}$, $\alpha'' = \alpha_n^{L''}$, $\beta' = \beta_n^{L'}$, $\beta'' = \beta_n^{L''}$;
    $h_n^\alpha \leftarrow h^\alpha$ and $h_n^\beta \leftarrow h^\beta$;
  for *each flow* f **do**
    // see Section 6.4.1.2
    $s_f \leftarrow$ sink of flow f;
    $h_f^\alpha \leftarrow \bar{\tau}_{f,s_f}^{L''} + \max_{n\in\text{path}(f)} h_n^\alpha$;
    // see (6.9) in Section 6.4.1.3
    $\alpha_f^{0,\text{UA}} \leftarrow \text{uaArrivalCurve}\left(\alpha_f^{0,\text{UPP}},\alpha_f^{0,L''},h_f^\alpha\right)$;
  for *each node* n **do**
    // see (6.10) in Section 6.4.1.3
    $\beta_n^{\text{UA}} \leftarrow \text{uaServiceCurve}\left(\beta_n^{\text{UPP}},\beta_n^{L''},h_n^\beta\right)$;
  $(\bar{d},\bar{\tau}) \leftarrow \text{GFP-TFA}\left(\alpha^{0,\text{UA}},\beta^{\text{UA}},E^{\text{cut}}\right)$;
else
  $(\bar{d},\bar{\tau}) \leftarrow \text{GFP-TFA}\left(\alpha^{0,\text{UPP}},\beta^{\text{UPP}},E^{\text{cut}}\right)$;
return $(\bar{d},\bar{\tau})$;

6.4.1.2 Sufficient Horizons for Arrival Curves of Flows at the Source

Consider flow f. As the arrival curve of flow f increases along its path, this horizon should be large enough such that the arrival curve of the flow, respects the compact domain at every node in its path. Specifically, consider node n in the path of flow f. Then, at the input of node n, an arrival curve for flow f is its arrival curve at the source, but shifted to the left by a delay-jitter bound from the source to node n, say $\tau_{f,n}^{\text{UPP}}$. Hence, sufficient horizon of flow f should be larger than or equal to $h_n^\alpha + \tau_{f,n}^{\text{UPP}}$ at every node n in its path. Thus, we use $h_f^\alpha = \max_{n \in \text{path}(f)} h_n^\alpha + \bar{\tau}_{f,s_f}^{L''}$ where s_f is the sink of flow f; note that $\bar{\tau}_{f,s_f}^{L''}$ is an upper-bound on the the end-to-end delay-jitter, for flow f, hence $\bar{\tau}_{f,s_f}^{L''} \geq \tau_{f,n}^{\text{UPP}}$.

Observe that the already computed compact domain h_n^β is a sufficient horizon for the service curve of node n.

6.4.1.3 Construction of UA Curves

Consider flow f. Function $\alpha_f^{0,\text{UA}} = \text{uaArrivalCurve}\left(\alpha_f^{0,\text{UPP}}, \alpha_f^{0,L''}, h_f^\alpha\right)$, at line 18 of Algorithm 6.3, constructs this UA curve as follows (see Fig. 6.4.1):

$$\alpha_f^{0,\text{UA}}(t) = \begin{cases} \alpha_f^{0,\text{UPP}}(t) & \text{if } t \leq h_f^\alpha \\ \alpha_f^{0,L''}(t) & \text{otherwise} \end{cases} \tag{6.9}$$

Consider node n. Function $\beta_n^{\text{UA}} = \text{uaServiceCurve}\left(\beta_n^{\text{UPP}}, \beta_n^{L''}, h_n^\beta\right)$, at line 20 of Algorithm 6.3, constructs this UA curve as follows (see Fig. 6.4.1):

$$\beta_n^{\text{UA}}(t) = \begin{cases} \beta_n^{\text{UPP}}(t) & \text{if } t \leq h_n^\beta \\ \min\left(\beta_n^{\text{UPP}}(h_n^\beta), \beta_n^{L''}(t)\right) & \text{otherwise} \end{cases} \tag{6.10}$$

6.4.2 Validity and Accuracy of FH-TFA

Theorem 6.4 (Validity and Accuracy of FH-TFA)**.** *Consider a FIFO network, as described in Section 6.2 and Algorithms 6.2 and 6.3. Then, (1) FH-TFA provides valid bounds. (2) If original, UPP service curves of all nodes are super-additive, FH-TFA and GFP-TFA provide the exact same bounds.*

The proof is in Section 6.6.3. In the common case where service curves are super-additive, FH-TFA and GFP-TFA return the same output, i.e., FH-TFA returns finite bounds if and only if GFP-TFA returns finite bounds and, if bounds are finite, both algorithms provide the same bounds. FH-TFA is thus a practical alternative to GFP-TFA, which may become too complex when there are many UPP curves.

FH-TFA is always applicable and provides valid bounds, as stated in item (1) in the theorem, and super-additivity of service curves is not a requirement for FH-TFA; furthermore, the obtained bounds by FH-TFA are guaranteed to be less than or equal to those obtained by FP-TFA (which uses linear curves). We use super-additivity of service curves only to prove that bounds obtained by FH-TFA are exactly equal to those of GFP-TFA. When service curves are not super-additive, it is not clear whether they are equal to those that would be obtained by GFP-TFA (using original, UPP curves), and this is left

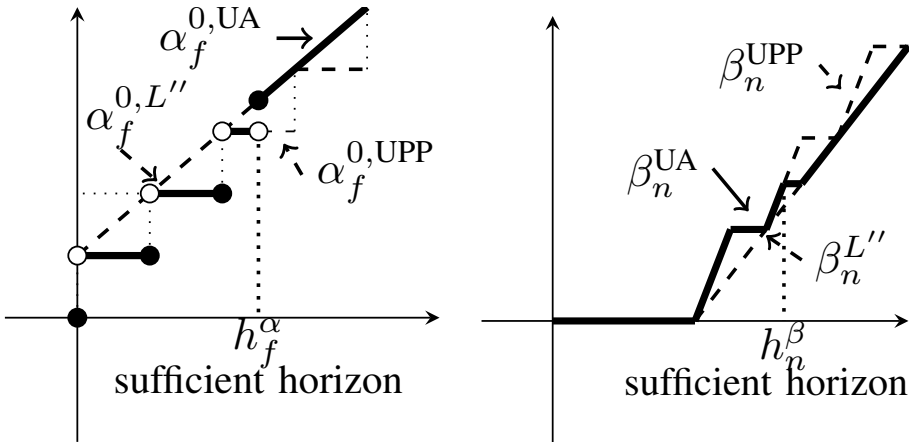


Figure 6.4.1: $\alpha_f^{0,\mathrm{UPP}}$ (resp. β_n^{UPP}) is the original, UPP arrival curve for flow f at the source (resp. service curve of node n), $\alpha_f^{0,L''}$ (resp. $\beta_n^{L''}$) is the smallest token-bucket (resp. largest rate-latency) that upper bounds $\alpha_f^{0,\mathrm{UPP}}$ (resp. lower bounds β_n^{UPP}), and h_f^{α} (resp. h_n^{β}) is a sufficient horizon for the arrival curve of flow f (resp. service curve of node n). Then, $\alpha^{0,\mathrm{UA}}$ (resp. β_n^{UA}) follows $\alpha_f^{0,\mathrm{UPP}}$ (resp. β_n^{UPP}) in $[0, h_f^{\alpha}]$ (resp. $[0, h_n^{\beta}]$) and beyond that follows $\alpha_f^{0,L''}$ (resp. $\beta_n^{L''}$).

for further study. Note that service curves of frequent schedulers, including DRR, WRR, IWRR, CBS, and Non-Preemptive Strict-Priority, are super-additive.

Remarks on time and space complexity: GFP-TFA iteratively calls function FF-TFA (line 5 of Algorithm 6.2) and the number of iterations cannot be determined in advance, however, we analyze the complexity of one instance of FF-TFA. FF-TFA (Algorithm 6.1) includes operations such as addition, min-plus convolution, horizontal deviation, etc. on UPP or UA functions; [164] formally studies the computational complexity of such operations, the addition being the most costly [164, Proposition 10]. FF-TFA calls once function aggragteArrivalCurve$_n$ for each node n (line 6 of Algorithm 6.3). With the addition being the most costly, the complexity of aggragteArrivalCurve$_n$ is in the order of the complexity of summing arrival curves of all flows. Specifically, assume that there are F flows with UPP arrival curves $\alpha_f^{0,\mathrm{UPP}}$ with a pseudo-period p_f and a rank T_f for $f \in [1,F]$ (see Section 6.1.1.1). Let p be the hyper-period of the aggregate and $T = \max_{f\in[1,F]} T_f$, and let M_f be the number of segments required to define $\alpha_f^{0,\mathrm{UPP}}$ in $[0, T+p)$ for $f \in [1,F]$. Then, by [164], the required space to compute $\sum_{f=1}^{F} \alpha_f^{0,\mathrm{UPP}}$ is $\sum_{f=1}^{F} M_f$ and the addition can be computed in time $O((\sum_{f=1}^{F} M_f)\log_2 F)$. Thus, as we have N nodes, one instance of FF-TFA inside GFP-TFA (using UPP curves) is run in time $O(N(\sum_{f=1}^{F} M_f)\log_2 F)$; the required space for GFP-TFA is $O(N(\sum_{f=1}^{F} M_f)\log_2 F)$ plus the space required to store service curves β_n^{UPP} for all node n.

However, FH-TFA restricts UPP functions to a finite horizon (i.e., UA functions) and applies GFP-TFA with UA curves. Specifically, it applies GFP-TFA with $\alpha_f^{0,\mathrm{UA}}$ defined in (6.9). Let M_f^h be the number of segments required to define $\alpha_f^{0,\mathrm{UPP}}$ in $[0, h_f^{\alpha})$ (i.e., the number of segments required to define $\alpha_f^{0,\mathrm{UA}}$; see Figure 6.4.1) for $f \in [1,F]$. With the same reasoning as the previous paragraph, one instance of FF-TFA inside GFP-TFA, using UA curves, is run in time $O(N(\sum_{i=1}^{F} M_f^h)\log_2 F)$; the required space for FH-TFA

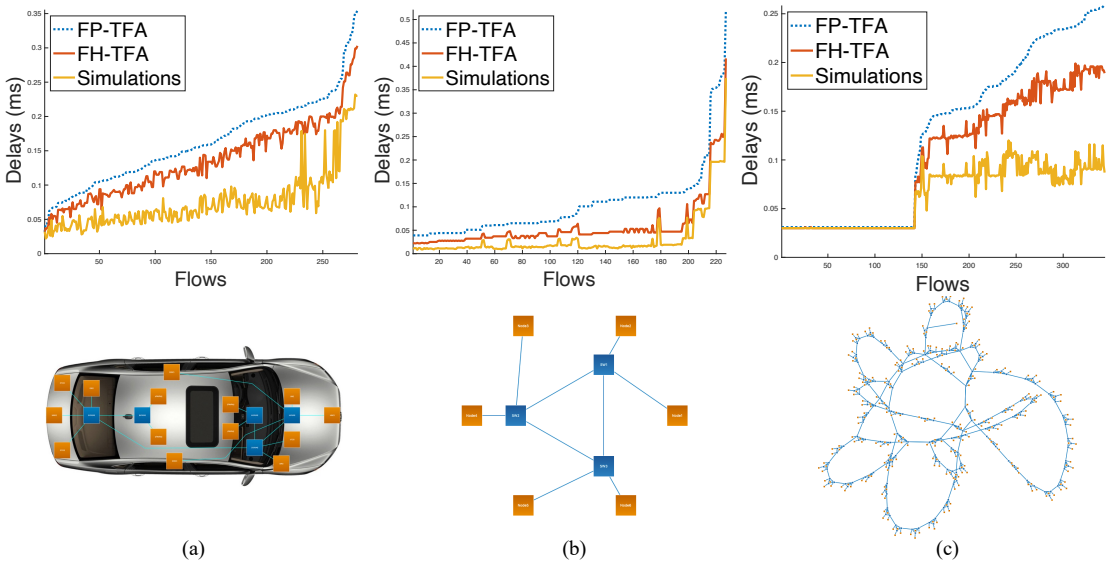


Figure 6.5.1: Delay bounds obtained with FP-TFA, which uses token-bucket arrival curves, and our FH-TFA. GFP-TFA with original UPP curves, fails as we face a memory size error. FH-TFAsignificantly improves the bounds compared to those obtained by FP-TFA. Moreover, the improvement is more considerable when we compare the tightness gaps using the simulation values. See Table 6.5.1 for run-times. Flows are ordered by values of FP-TFA.

(GFP-TFA using UA curves) is $O(N(\sum_{i=1}^{F} M_f^h)\log_2 F)$ plus the space required to store service curves β_n^{UA} for all node n. As the number of segments in the horizon, $\sum_{f=1}^{F} M_i^h$ (i.e., transient part) might be extremely smaller than those of the hyper-period, $\sum_{f=1}^{F} M_f$ (see Figure 6.2.1 where $\sum_{f=1}^{F} M_i^h = 13$ and $\sum_{f=1}^{F} M_f = 2020$.), FH-TFA might significantly reduce the time and space complexity compared to GFP-TFA. As observed in Section 7.5, GFP-TFA has high computational complexity and is an impractical method, while FH-TFA is successfully applied to each of our industrial case studies.

6.5 Numerical Evaluation

We use three real, industrial networks provided by industrial partners of RTaW, a leading company in Ethernet TSN design, performance evaluation and automated configuration tools; these examples are anonymized and slightly changed. We first explain each network, and we then present the results.

6.5.1 A Feed-Forward Network

It is illustrated in Fig. 6.5.1 (a): This network is feed-forward. It has 13 end-nodes and 5 switches: Link speeds are $c = 2$Gb/s and switches (blue squares) have switching latency equal to $L = 15\mu$s, i.e., nodes offer a rate-latency service curve $\beta_{c,L}$ with $L = 0$ for end-nodes and $L = 15$ for switches. There are 50 periodic flows: periods are 0.03, 0.06, 0.12, 0.24, 1, 5, 10, 20, 25, 60, 125, 200, and 1000 ms; packet sizes are [130, 1360] bytes.

6.5.2 A Small-sized Network with Cyclic Dependencies

It is illustrated in Fig. 6.5.1 (b): This network has cyclic dependencies. It has 6 end-nodes and 3 switches: Link speeds are $c = 2$Gb/s and switches (blue squares) have switching latency equal to $L = 1.5\mu$s, i.e., nodes offer a rate-latency service curve $\beta_{c,L}$ with $L = 0$ for end-nodes and $L = 1.5$ for switches. There are 56 periodic flows: periods are 0.24, 0.25, 0.28, 0.5, 0.8, 1, 1.28, 2, 2.4, 20, 24, 100, 800, 1600, and 2400 ms; packet sizes are [65,530] bytes.

6.5.3 An Extremely Large Network with Cyclic Dependencies

It is illustrated in Fig. 6.5.1 (c): This network has cyclic dependencies. It has 291 end-nodes and 220 switches: Link speeds are $c = \{0.1, 1, 10\}$Gb/s and switches (blue squares) have switching latency equal to $L = 2\mu$s. There are 486 periodic flows: periods are 0.125, 0.24, 0.28, 0.608, and 10 ms; packet sizes are [96,1518] bytes.

6.5.4 Results

We apply three methods to compute end-to-end delay bounds: 1) GFP-TFA with original UPP curves, i.e., stair arrival curve for flows (see Fig. 6.1.1); 2) FP-TFA; 3) FH-TFA. Also, we use the RTAW-Pegase tool to do simulations where we compute some true delays for each flow that serve as lower-bounds on the worst-case. Note that the true worst-case is between the simulation bound and the smallest delay bound, hence this provides a bound on the tightness.

First, we observe that in all three networks, GFP-TFA with original UPP curves, fails as we face a memory size error. Indeed as explained before, GFP-TFA has high computational complexity and is an impractical method that is used to validate our second version of TFA, FH-TFA. Second, FP-TFA, which uses token-bucket arrival curves, and FH-TFA are successfully applied to each network, and obtained delay bounds are illustrated in Fig. 6.5.1: FH-TFA significantly improves the bounds compared to those obtained by FP-TFA with token-bucket arrival curves; namely, FH-TFA improves bounds by around 20% (median) and 65% (maximum) compared to FP-TFA in our examples. This increases efficiency as more traffic can be accepted while meeting required deadlines, and making the network more robust to changes in network conditions and physical infrastructure. Moreover, the improvement is more considerable when we compare the tightness gap using the simulation values. Note that as service curves are rate-latency, hence super-additive, by Theorem 6.4, FH-TFA and GFP-TFA provide the exact same bounds.

Table 6.5.1: Run-times for networks of Fig. 6.5.1

Method	Network (a)	Network (b)	Network (c)
GFP-TFA with UPP curves	-	-	-
FP-TFA (s)	[0.053, 0.059]	[2.66, 2.68]	[10.8, 10.88]
FH-TFA (s)	[0.8, 0.82]	[7.07, 7.1]	[35.17, 35.37]

We also provide run-times in Table 6.5.1: We use the min-plus interpreter of RTaW [92] that has infinite precision using rational numbers. We use Java on a 2.6 GHz 6-Core Intel Core i7 computer. As GFP-TFA with UPP curves fails hence no run-time is provided; for the other methods, we run the program ten times, and 95% confidence interval is reported. FH-TFA is fast and practical, even for the extremely large network. Note that, in our experiment, we use the minimum resolution Δ equal to 1 nanosecond

(line 6 of Algorithm 6.2).

6.6 Proofs

6.6.1 Proof of Theorem 6.2

Proof of Theorem 6.2. As the packetizer is fed by a transmission link with rate c, it follows that when a packet of size l arrives to the packetizer, it is released after a time equal to $\frac{l}{c}$. Hence, the release time of any packet is upper-bounded by $\frac{l^{\max}}{c}$. Then, by [40, Theorem 6.2], it follows that a pure delay $\delta_{\frac{l^{\max}}{c}}$ is a service curve for the packetizer, hence the output arrival curve is min-plus deconvolution of the input arrival curve and $\delta_{\frac{l^{\max}}{c}}$. □

6.6.2 Proof of Theorem 6.3

Proof of Theorem 6.3. The proof follows similar steps as in the proof of Theorem 2 of [119]. Let F be the mapping that maps $\tau^{\text{cut},k-1}$ to $\tau^{\text{cut},k}$ in lines 1-6 of Algorithm 6.2.

Fix an acceptable trajectory scenario, i.e., the set of all cumulative arrival functions at all nodes in the networks such that arrival curve and service curve constraints are met. Let t^{prop} be the minimum of all link propagation delays and $\theta = \frac{t^{\text{prop}}}{2}$, so that $0 < \theta < t^{\text{prop}}$. Consider an arbitrary $\eta \geq 0$.

- Let W (resp. U) represents the set of point located just before (resp. just after) the cuts. Note that in the original, uncut network, U and W are connected via some links. Also, the network between U and W is feed-forward.
- Let V represent the points located exactly θ seconds before W. As $\theta < t^{\text{prop}}$ and t^{prop} is the minimum propagation delay, V is on the same links as W.
- For $M = \{U, V, W\}$, let C_M represent the collection of cumulative arrival functions; let C_M^{η} represent the collection of cumulative arrival functions stopped at time η, i.e., $C_M^{\eta}(t) = \min\big(C_M(t), C_M(\eta)\big)$; let $C_M^{'\eta}$ represent the collection of cumulative arrival functions when inputs at U and all sources are stopped at time η.
- For $M = \{U, V, W\}$, we denote by τ_M the collection of worst-case delay jitter from the source to M for all flows at M; we denote by τ_M^{η} collection of worst-case delay jitter from the source to M for all flows at M observed up to time η; we denote by $\tau_M^{'\eta}$ collection of worst-case delay jitter from the source to M for all flows at M when inputs at U and all sources are stopped at time η.

First, observe that τ_M^{η} and $\tau_M^{'\eta}$ are finite. This is because as η is finite and as sources are constrained at the source, a finite number of bits ever entered the network, hence delays are finite. Then:

1) observe that $\tau_V^{\eta} \leq \tau_V^{'\eta}$; This is implied by the fact that the network is causal, and hence, $\forall\, t \leq \eta, C_M^{'\eta}(t) = C_M^{\eta}(t)$ and $\forall\, t > \eta, C_M^{'\eta}(t) \geq C_M^{\eta}(t)$.

2) As the network between U and W is feed-forward, function F computes a bound on the delay-jitters from the source to W, given delay-jitters from the source to U, and hence, $\tau_W^{'\eta} \leq F\big(\tau_U^{\eta}\big)$.

3) As there is a constant delay θ between V and W, $\forall\, t \geq 0,\ C'^{\eta}_{W}(t+\theta) = C'^{\eta}_{V}(t)$; it follows that $\tau'^{\eta}_{W} \leq F\left(\tau^{\eta}_{U}\right)$ $\tau'^{\eta}_{V} = \tau'^{\eta}_{W}$.

Combine **1)**, **2)**, and **3)** and obtain

$$\tau^{\eta}_{V} = F\left(\tau^{\eta}_{U}\right) \tag{6.11}$$

Observe that $\tau^{\eta}_{V} = \tau^{\eta+\theta}_{W}$. This is because the exact same traffic that is observed by V between 0 to η is observed by W between θ to $\eta+\theta$, and the difference is only a constant delay θ; this does not change the delay-jitter. Also, as in the original, uncut network U and W are connected, $\tau^{\eta}_{W} = \tau^{\eta+\theta}_{U}$. Thus, $\tau^{\eta}_{V} = \tau^{\eta+\theta}_{U}$. Combine this with (6.11) and obtain $\tau^{\eta+\theta}_{U} = F\left(\tau^{\eta}_{U}\right)$. This is valid for every $\eta \geq 0$, apply it with $\eta = k\theta$ and obtain: $\forall\, k \geq 0, \tau^{(k+1)\theta}_{U} = F\left(\tau^{k\theta}_{U}\right)$. As the network is empty at time zero, $\tau^{0}_{U} = 0$. As F is wide-sense increasing and $F(\bar{\tau}) = \bar{\tau}$ and a simple induction, it follows that $\tau^{k\theta}_{U} \leq \bar{\tau}$ and hence $\tau^{\eta}_{U} \leq \bar{\tau}$ for $\eta \geq 0$. Hence, $\sup_{\eta \geq 0} \tau^{\eta}_{U} \leq \bar{\tau}$, i.e., for any acceptable trajectory scenario, the worst-case delay jitter bound from a source to a cut for every flow at cuts is upper-bounded by $\bar{\tau}$. □

6.6.3 Proof of Theorem 6.4

Proof of Theorem 6.4. The validity of the bounds is directly implied by the validity of the constructed UA curves. Specifically, UA curves constructed in FH-TFA, are safe upper/lower bounds of the original, UPP curves, i.e., $\alpha^{0,\mathrm{UPP}} \leq \alpha^{0,\mathrm{UA}}$ and $\beta^{\mathrm{UA}} \leq \beta^{\mathrm{UPP}}$ (see Fig. 6.4.1). Then, as GFP-TFA is isotone, it follows that bounds obtained by FH-TFA are larger than or equal to those obtained by GFP-TFA (using original, UPP curves), hence valid and (1) is shown.

We now proceed to show item (2) when service curves are super-additive. We first prove the following lemmas.

Lemma 6.1. *Consider Algorithm 6.3 and consider node n. Then, the computation of the delay bound at node n, $d^{UPP}_{n} = hDev\left(\alpha^{UPP}_{n} - l^{min}_{n}, \beta^{UPP}_{n}\right) + \frac{l^{min}_{n}}{c_n}$ (see line 8 of Algorithm 6.1), involves only values of $\alpha^{UPP}_{n}(t)$ for $t \in \left[0, h^{\alpha}_{n}\right]$ and $\beta^{UPP}_{n}(t)$ for $t \in \left[0, h^{\beta}_{n}\right]$, where α^{UPP}_{n} is the aggregate arrival curve at node n computed by GFP-TFA using UPP curves, l^{min}_{n} is the minimum packet size of flows at node n, and c_n is the transmission rate at node n.*

Proof. As $\alpha^{0,L''}_{f} \geq \alpha^{0,\mathrm{UPP}}_{f}$ (resp. $\alpha^{0,L'}_{f} \leq \alpha^{0,\mathrm{UPP}}_{f}$) and $\beta^{L''}_{n} \leq \beta^{\mathrm{UPP}}_{n}$ (resp. $\beta^{L'}_{n} \geq \beta^{\mathrm{UPP}}_{n}$) and as GFP-TFA is isotone, it follows that $(d^{L'}, \tau^{L'}) \leq (d^{\mathrm{UPP}}, \tau^{\mathrm{UPP}}) \leq (d^{L''}, \tau^{L''})$. Thus, $\alpha^{L'}_{n} \leq \alpha^{\mathrm{UPP}}_{n} \leq \alpha^{L''}_{n}$. Also, by construction, $\beta^{L''}_{n} \leq \beta^{\mathrm{UPP}}_{n} \leq \beta^{L'}_{n}$. Apply Theorem 6.1 with $\alpha = \alpha^{\mathrm{UPP}}_{n}$, $\alpha'' = \alpha^{L''}_{n}$, $\alpha' = \alpha^{L'}_{n}$, $\beta = \beta^{\mathrm{UPP}}_{n}$, $\beta'' = \beta^{L''}_{n}$, and $\beta' = \beta^{L'}_{n}$ to conclude that $\mathrm{hDev}\left(\alpha^{\mathrm{UPP}}_{n}, \beta^{\mathrm{UPP}}_{n}\right)$ depends only on values of $\alpha^{\mathrm{UPP}}_{n}(t)$ for $t \in \left[0, h^{\alpha}_{n}\right]$ and $\beta^{\mathrm{UPP}}_{n}(t)$ for $t \in \left[0, h^{\beta}_{n}\right]$. Observe that these compact domains are also sufficient for the computation of $\mathrm{hDev}\left(\alpha^{\mathrm{UPP}}_{n} - l^{\min}_{n}, \beta^{\mathrm{UPP}}_{n}\right)$. □

Lemma 6.2. *Consider Algorithm 6.3 and assume a τ^{cut} such that $0 \leq \tau^{cut} \leq \bar{\tau}^{cut,L''}$ where $\bar{\tau}^{cut,L''}$ are delay-jitters for flows at cuts extracted from $\bar{\tau}^{L''}$, obtained at line 7. Then, FF-TFA$\left(\alpha^{0,UPP}, \beta^{UPP}, E^{cut}, \tau^{cut}\right)$ = FF-TFA$\left(\alpha^{0,UA}, \beta^{UA}, E^{cut}, \tau^{cut}\right)$ where FF-TFA is described in Algorithm 6.1.*

Proof. First, we show that for every node n,

$$\forall\, t \in [0,\ h^{\beta}_{n}],\ \beta^{\mathrm{UA}}_{n}(t) = \beta^{\mathrm{UPP}}_{n}(t) \tag{6.12}$$

$$\forall f \in \text{flows}(n),\ \forall t \in [0,\ h_n^\alpha],\ \alpha_{f,n}^{\text{UA}}(t) = \alpha_{f,n}^{\text{UPP}}(t) \tag{6.13}$$

$$d_n^{\text{UA}} = d_n^{\text{UPP}} \tag{6.14}$$

Observe that (6.12) is by construction. We now prove (6.13) and (6.14). by induction on node n. Recall that, as explained in Section 6.2, nodes are labeled in a topological order of the cut network (such orders exist as the cut network is feed-forward). We assume, to simplify the notation, that the node label n is an integer that reflects this topological order. The base case of our induction is thus for a node n that is an edge node, i.e., where all flows at node n are either fresh or cut.

Base Case: n is an edge node in the cut network

f **is fresh**: We show that for every fresh f, $\forall t \in [0,\ h_n^\alpha],\ \alpha_f^{0,\text{UA}}(t) = \alpha_f^{0,\text{UPP}}(t)$. By construction, $\alpha_f^{0,\text{UA}}(t) = \alpha_f^{0,\text{UPP}}(t)$ for $t \in [0,\ h_f^\alpha]$, hence we should show that $h_f^\alpha \geq h_n^\alpha$: Recall that $h_f^\alpha = \max_{n \in \text{Path}(f)} h_n^\alpha + \bar{\tau}_{f,s_f}^{L''}$, and as $\bar{\tau}_{f,s_f}^{L''} \geq 0$, it follows that $h_f^\alpha \geq h_n^\alpha$.

f **is cut**: For every flow f that is cut at node n (if any), we show that $\forall t \in [0,\ h_n^\alpha],\ \alpha_f^{0,\text{UA}}\left(t + \tau_f^{\text{cut}}\right) = \alpha_f^{0,\text{UPP}}\left(t + \tau_f^{\text{cut}}\right)$. By construction, $\alpha_f^{0,\text{UA}}(t) = \alpha_f^{0,\text{UPP}}(t)$ for $t \in [0,\ h_f^\alpha]$, hence we must show that $h_f^\alpha \geq h_n^\alpha + \tau_{f,n}^{\text{cut}}$: Recall that $h_f^\alpha = \max_{n \in \text{Path}(f)} h_n^\alpha + \bar{\tau}_{f,s_f}^{L''}$; observe that $h_f^\alpha \geq h_n^\alpha + \bar{\tau}_{f,s_f}^{L''}$. Then, as $s_f \geq n$ and as delay-jitter increases along the path, $\bar{\tau}_{f,s_f}^{L''} \geq \bar{\tau}_{f,n}^{L''}$, hence $h_f^\alpha \geq h_n^\alpha + \bar{\tau}_{f,n}^{L''}$. By construction, as $\bar{\tau}^{\text{cut},L''}$ is extracted form $\bar{\tau}^{L''}$, we have $\bar{\tau}_{f,n}^{L''} = \bar{\tau}_{f,n}^{\text{cut},L''}$, hence $h_f^\alpha \geq h_n^\alpha + \bar{\tau}_{f,n}^{\text{cut},L''}$. Lastly, by hypothesis of the lemma, $\bar{\tau}^{\text{cut},L''} \geq \tau^{\text{cut}}$, and therefore, $h_f^\alpha \geq h_n^\alpha + \tau_{f,n}^{\text{cut}}$.

Hence, the base case is shown for (6.13). Then, by Lemma 6.1, (6.12), and (6.13), it follows that $d_n^{\text{UA}} = d_n^{\text{UPP}}$ hence the base case is shown for (6.14).

Induction Case: n is not an edge node in the cut network

In this case, we assume that for every $n' < n$, (6.13) and (6.14) holds, and we prove them at node n. First, observe that for a fresh flow or cut flow $f \in \text{flows}(n)$, the proof of (6.13) is similar to the base case. Second, for a transit flow $f \in \text{flows}(n)$, (6.13) can be rewritten as follows:

$$\forall t \in [0,\ h_n^\alpha],\ \alpha_f^{0,\text{UA}}(t + \tau_{f,n}^{\text{UA}}) = \alpha_f^{0,\text{UPP}}(t + \tau_{f,n}^{\text{UPP}}) \tag{6.15}$$

By induction hypothesis for (6.14), we have $d_{n'}^{\text{UA}} = d_{n'}^{\text{UPP}}$ for $n' < n$, thus $\tau_{f,n}^{\text{UA}} = \tau_{f,n}^{\text{UPP}}$ is implied by the construction. Hence, (6.15) can be rewritten as follows:

$$\forall t \in [0,\ h_n^\alpha],\ \alpha_f^{0,\text{UA}}(t + \tau_{f,n}^{\text{UPP}}) = \alpha_f^{0,\text{UPP}}(t + \tau_{f,n}^{\text{UPP}}) \tag{6.16}$$

Recall that $\alpha_f^{0,\text{UA}}(t) = \alpha_f^{0,\text{UPP}}(t)$ for $t \in [0,\ h_f^\alpha]$, we need to show that $h_f^\alpha \geq \tau_{f,n}^{\text{UPP}} + h_n^\alpha$. By construction, $h_f^\alpha = \max_{n \in \text{Path}(f)} h_n^\alpha + \bar{\tau}_{f,s_f}^{L''} \geq h_n^\alpha + \bar{\tau}_{f,s_f}^{L''}$. Observe that $\bar{\tau}_{f,s_f}^{L''} \geq \bar{\tau}_{f,n}^{L''}$ and $\bar{\tau}_{f,n}^{L''} \geq \tau_{f,n}^{\text{UPP}}$, and hence $h_f^\alpha \geq h_n^\alpha + \tau_{f,n}^{\text{UPP}}$. This shows (6.15) and hence the induction case is shown for (6.13). Then, by Lemma 6.1, (6.12), and (6.13), it follows that $d_n^{\text{UA}} = d_n^{\text{UPP}}$ hence the induction case is shown for (6.14). Therefore, (6.13) and (6.14) are shown. Lastly, by (6.14), both methods return the same collection of per-node delay bounds, hence also of delay-jitters for flows. □

We now proceed to conclude the proof of item (2) of Theorem 6.4. Observe that if one of the components

of $(\bar{d}^{L''}, \bar{\tau}^{L''})$, obtained in line 7 of Algorithm 6.3, is infinite, then FH-TFA applies GFP-TFA with the original, UPP curves hence (2) is concluded. We now proceed by assuming that $(\bar{d}^{L''}, \bar{\tau}^{L''})$ is finite (thus the network is stable) and show that GFP-TFA$\left(\alpha^{0,\mathrm{UA}}, \beta^{\mathrm{UA}}, E^{\mathrm{cut}}\right)$ returns the same bounds as GFP-TFA$\left(\alpha^{0,\mathrm{UPP}}, \beta^{\mathrm{UPP}}, E^{\mathrm{cut}}\right)$. First, observe that GFP-TFA starts with $\tau^{\mathrm{cut}} = 0$. Next, as FF-TFA is isotone, bounds obtained at each iteration inside GFP-TFA, using UA or UPP curves, are upper-bounded by those obtained using linear curves; hence, for flows at cuts, delay-jitter bounds obtained at each iteration using UA or UPP curves are always upper-bounded by the fix-point $\bar{\tau}^{\mathrm{cut},L''}$. Then, iteratively apply Lemma 6.2 to conclude that at each iteration where GFP-TFA calls FF-TFA the same bounds are obtained using UA and UPP curves. Therefore, FH-TFA and GFP-TFA provide the exact same finite bounds. □

6.7 Conclusion

TFA quickly becomes intractable when there is a large number of periodic sources and UPP curves. This problem is frequently observed in time-sensitive and real-time networks. We provide a practical solution, FH-TFA, that obtains delay bounds that are formally proven, and that can handle a large number of periodic sources with different periods.

6.8 Notation

Table 6.8.1: Notation List, Specific to Chapter 6

f	A flow
E^{cut}	Cutset: removing E^{cut} creates a feed-forward graph
$\mathcal{G} = (\mathcal{N}, \mathcal{E})$	The graph induced by flows of class
path(f)	The sequence of output ports in the path of flow f
$\mathcal{N}, n$	The set of all output ports, an output port
$\mathcal{E}, e$	The set of edges, an edge
In(n)	The set of edges of that are incidents at node n
flows(n)	The set of flows at output port n
α_n	Aggregate arrival curve of all flows at the input of node n
α_n^{fresh}	Aggregate arrival curve of all fresh flows at the input of node n
$\alpha_n^{\text{transit}}$	Aggregate arrival curve of all transit flows at the input of node n
$\alpha_{f,n}$	An arrival curve for flow f at the input of node n
α_f^0	An arrival curve for flow f at the source
β_n	A service curve offered to node n
α^0	Collection of arrival curves of all flows at the source
d	Collection of delay bound d_n for every node n
τ^{cut}	Collection of delay-jitter bounds $\tau_{f,n}$ for every flow f at cuts
τ	Collection of delay-jitter bounds $\tau_{f,n}$ for every flow f at every node n in its path
β	Collection of service curves of all nodes
d_n	Delay bound at node n
$\tau_{f,n}$	Delay-jitter bound for flow f from the source to the input of node n
$l_f^{\max}$	Maximum packet size for flow f
$l_f^{\min}$	Minimum packet size for flow f
Δ	Minimum resolution of times, e.g., 1 nanosecond
h	Sufficient horizon
c_n	Transmission rate of the line fed by output port n
δ_d	Pure delay function with $\delta_d(t) = 0$ for $t \le d$ and $\delta_d(t) = \infty$ for $t > d$
$\beta_{c,L}$	Rate-latency function with $\beta_{c,L}(t) = \max(0, c(t-L))$
$\mathbb{Q}^+$	Set of non-negative rational numbers
$\mathcal{F}^{\text{piece-wise-linear}}$	Set of piece-wise linear and wide-sense increasing functions $f : \mathbb{Q}^+ \mapsto \mathbb{Q}^+ \cup \{+\infty\}$
$\nu_{p,b}$	Stair function with $\nu_{p,b}(t) = b\left\lceil \frac{t}{p} \right\rceil$
$\gamma_{r,b}$	Token-bucket function with $\gamma_{r,b}(0) = 0$ and $\gamma_{r,b}(t) = r\,t + b$ for $t > 0$
hDev	Horizontal deviation $\text{hDev}(\alpha, \beta) = \sup_{t \ge 0}\{\inf\{d \ge 0 \mid \alpha(t) \le \beta(t+d)\}\}$
$\otimes$	Min-plus convolution $(f \otimes g)(t) = \inf_{0 \le s \le t}\{f(t-s) + g(s)\}$
$\oslash$	Min-plus deconvolution $(f \oslash g)(t) = \sup_{s \ge 0}\{f(t+s) - g(s)\}$
vDev	Vertical deviation $\text{vDev}(\alpha, \beta) = \sup_{t \ge 0}\{\alpha(t) - \beta(t)\}$

7 Quasi-Deterministic Burstiness Bound for Aggregate of Independent, Periodic Flows

In time-sensitive networks, where moments unfold,
Monitoring the status, a story to be told.
Devices send packets, with periodic grace,
Aggregated and forwarded, to the controller's embrace.

Bounding aggregate burstiness, a task at hand,
For effective resource management, a demand so grand.
Independent and periodic flows, our focus true,
Bounding their burstiness, with insights anew.

Deterministic bounds, in perfect sync they lie,
But in practice, unlikely, as time passes by.
Overly pessimistic, an impractical view,
Probability emerges, shedding light so true.

For flows synchronized, with periods aligned,
A closed-form bound, a treasure we find.
Dvoretzky-Kiefer-Wolfowitz, inequality's name,
A bound that shines, in mathematical fame.

Heterogeneous realms, where diversity thrives,
Grouping flows, through creative strives.
Convolution bound, the bounds we combine,
For aggregate burstiness, a path so fine.

Numerical proximity, simulations reveal,
Tight bounds obtained, their accuracy we feel.
Aggregate burstiness, with a non-zero chance,
Smaller than deterministic, in a graceful dance.

Growth transformed, as n takes its stand,
$\sqrt{n \log n}$, a growth that feels grand.
For number of flows, an elegant decree,
Burstiness estimation, in mathematical glee.

So let this chapter commence, a journey we embark,
In time-sensitive networks, where resource management sparks.
Bounding aggregate burstiness, with precision and might,

Chapter 7. Quasi-Deterministic Burstiness Bound for Aggregate of Independent, Periodic Flows

A tale of bounds, shining with insightful light.

Created with ChatGPT, free research preview (version May 24) [141]

We already proposed a solution that more accurately handles periodic flows by using UPP curves, and we mitigated the tractability issue of aggregating many UPP curves. As explained in Section 1.3.2, an orthogonal direction to reduce the pessimism of aggregating arrival curve constraints is to use the affine functions but to permit some violation probability. The development of industrial automation requires timely and accurate monitoring of the status of the network. In time-sensitive networks, a common assumption for critical types of traffic is that devices send packets periodically. These packets are aggregated and forwarded to the controller. Characterizing this aggregate traffic is then crucial for effective resource management.

We consider independent, periodic flows and are interested in bounding the burstiness of their aggregate traffic. If all flows are synchronized (i.e., have the same phase), then the aggregate burstiness is the sum of the packet sizes of all flows. Otherwise, the aggregate burstiness would likely be smaller. We assume that phases are random and uniformly distributed, and we are then interested in finding the probability that the aggregate burstiness exceeds some pre-specified value, i.e., *bounds on the tail probability of the aggregate burstiness*. This enables us to estimate an upper bound for the aggregate burstiness, which is valid with probability of at least $1-\varepsilon$ throughout the entire network's lifetime, where ε is a small, non-zero violation tolerance. (Deterministic) network calculus can then be applied to obtain delay and backlog bounds that are valid with probability of at least $1-\varepsilon$ (*quasi-deterministic* bounds). As we show in Section 7.3, such quasi-deterministic bounds are considerably less than those obtained if we do not allow any probability of violation (deterministic bounds).

To overcome this issue, probabilistic versions of network calculus (known as *Stochastic Network Calculus*) have emerged, and their aim is to compute performances when a small *violation probability* is allowed. Using probabilistic tools such as moment-generating functions [97] or martingales [98], existing works [97, 99, 100, 101, 102] do not provide quasi-deterministic bounds, rather they find the probability of delay or backlog violation at an arbitrary point in time, in stationary regime; however, in time-sensitive networks, we are interested in the probability that a delay or backlog bound is not violated during some interval (e.g., the network's lifetime), not just one arbitrary point in time. The violation probability of a delay bound being small at one arbitrary point in time, does not imply that the probability that the delay bound is never violated during a period of interest is small. In fact, there would likely be some violations. [101, Section 4.4] points out that quasi-deterministic bounds are always trivial when arrival processes are stationary and ergodic. However, it has been overlooked that there is interest in some non-ergodic arrival processes, as in our case. Indeed, with our model, phases are drawn randomly but remain the same during the entire period of interest; thus, our arrival processes are not ergodic. As of today, we present the only known method that obtains quasi-deterministic bounds for independent, periodic flows.

Contributions of this chapter are the following:

1. For the homogeneous case (i.e., flows with the same packet size and period), we provide a closed-form expression that bounds the tail probability of the aggregate burstiness (Theorem 7.1). We obtain this bound using the Dvoretzky–Kiefer–Wolfowitz (DKW) inequality [170]. When the number of flows is large, this bound is fairly tight compared to simulations. Also, it results in a closed-form expression for the quasi-deterministic burstiness estimated for a non-zero violation probability (Corollary 7.1); it grows in $\sqrt{n \log n}$, unlike the deterministic one that grows in n,

where n is the number of flows. We also provide Theorem 7.2, a refinement of Theorem 7.1 that provides a slightly better bound when the number of flows is small, but at the expense of not having a closed-form expression.

2. For the heterogeneous case (i.e., flows with different packet sizes and periods), we obtain a bound by grouping flows into homogeneous sets and combining the bounds obtained for each set, using a convolution bounding technique based on Abel's summation (Theorem 7.3). The obtained convolution bound can be efficiently computed using discrete convolution. For a specific case, when flows have the same period and different packet sizes, we provide an alternative bound that may be better, when the number of flows per packet size is small (Theorem 7.4).

3. We numerically show that our bounds are close to simulations. The quasi-deterministic aggregate burstiness we obtain with a small, non-zero violation tolerance is considerably smaller than the deterministic one. For the heterogeneous case, we show that our convolution bounding technique provides bounds significantly smaller than those obtained by the union bound.

The rest of this chapter is organized as follows: We present our model in Section 7.1, and then in Section 7.2, we present some results from state-of-the-art. Our contributions are detailed in Section 7.3 for the homogeneous case and in Section 7.4 for the heterogeneous case. Finally, we provide some simulation results in Section 7.5 to demonstrate the tightness of the bounds. In Section 7.6, we conclude the chapter. A summary of notation and symbols used in this chapter are given in Section 7.7.

7.1 Assumptions and Problem Statement

In the whole chapter, we will denote $\mathbb{N} = \{0, 1, \ldots\}$ and $\mathbb{N}_n = \{1, \ldots, n\}$.

7.1.1 Assumptions

We consider n periodic flows of packets. Each flow $f \in \mathbb{N}_n$ is periodic with period τ_f and phase $\phi_f \in [0, \tau_f)$, and sends packets of size ℓ_f: the number of bits of flow f arriving in the time interval $[0, t)$ is $\ell_f \lceil [t - \phi_f]^+ / \tau_f \rceil$ where we use the notation $[x]^+ = \max(0, x)$ and $\lceil \rceil$ denotes the ceiling.

For every flow f, we assume that ϕ_f is random, uniformly distributed in $[0, \tau_f]$, and that the different $(\phi_f)_{f \in \mathbb{N}_n}$ are independent random variables.

7.1.2 Problem Statement

We consider the aggregation of the n flows and let $A[s, t)$ denote the number of bits observed in time interval $[s, t)$. Our goal is to find a token-bucket arrival curve constraining this aggregate, that is, a rate r and a burst b such that $\forall s \leq t,\ A[s, t) \leq r(t - s) + b$. It follows from the assumptions that each individual flow $f \in \mathbb{N}_n$ is constrained by a token-bucket arrival curve with rate $r_f = \ell_f / \tau_f$ and burst ℓ_f. Therefore, the aggregate flow is constrained by a token-bucket arrival curve with rate $r^{\text{tot}} = \sum_{f=1}^{n} r_f$ and burst $\ell^{\text{tot}} = \sum_{f=1}^{n} \ell_f$.

However, due to the randomness of the phases, ℓ^{tot} might be larger than what is observed, and we are rather interested in token-bucket arrival curves with rate r^{tot} and a burst b valid with some probability; specifically, we want to find a bound on the tail probability of the aggregate burstiness, which is defined

as the smallest value of B such that the aggregate flow is constrained by a token-bucket arrival curve with rate r^{tot} and burst B, for the entire network lifetime. The aggregate burstiness is given by

$$B = \sup_{t \geq 0} \bar{B}(t). \tag{7.1}$$

where $\bar{B}(t)$ is the token-bucket content at time t for a token-bucket that is initially empty, and is given by

$$\bar{B}(t) = \sup_{s \leq t} \{A[s,t) - r^{\text{tot}}(t-s)\}. \tag{7.2}$$

Note that B is a function of the random phases of the flows, therefore, is also random. Assume that $\mathbb{P}(B > b) = \varepsilon$; this means that, with probability $1-\varepsilon$, after periodic flows started, the aggregate burstiness is $\leq b$. Conversely, with probability ε, the aggregate burstiness is $> b$.

Observe that $\mathbb{P}(B > b) = 0$ for all $b \geq \ell^{\text{tot}}$, as ℓ^{tot} is a deterministic bound on the aggregate burstiness. Then, for some pre-specified value $0 \leq b < \ell^{\text{tot}}$, our problem is equivalent to finding $\epsilon(b)$ that bounds the tail probability of the aggregate burstiness B, i.e.,

$$\mathbb{P}(B > b) \leq \epsilon(b). \tag{7.3}$$

7.2 Related Works

Bounding the burstiness of flows in Network Calculus is an important problem since it has a strong influence on the delay and backlog bounds. The deterministic aggregate burstiness can be improved (compared with summing burstiness of all flows) when the phases of the flows are known exactly [171].

Regarding the *Stochastically Bounded Burstiness (SBB)* [101, 172], three models have been proposed, depending on how quantifiers are used,

$$\text{SBB}: \forall 0 \leq s \leq t,\ \mathbb{P}\left(A[s,t) - r(t-s) > b\right) \leq \epsilon(b), \tag{7.4}$$

$$\text{S}^2\text{BB}: \forall t \geq 0,\ \mathbb{P}\Big(\sup_{0 \leq s \leq t} \{A[s,t) - r(t-s)\} > b\Big) \leq \epsilon(b), \tag{7.5}$$

$$\text{S}^3\text{BB}: \mathbb{P}\Big(\sup_{0 \leq s \leq t} \{A[s,t) - r(t-s)\} > b\Big) \leq \epsilon(b). \tag{7.6}$$

First, notice that $\text{S}^3\text{BB} \implies \text{S}^2\text{BB} \implies \text{SBB}$. Indeed, SBB is a probability upper bound that the arrival curve constraint is invalid for a fixed pair of times $s \leq t$. In contrast, S^2BB is the probability that token-bucket content at time t, $\bar{B}(t)$ exceeds b, hence the "$\forall s$" appearing inside the probability. Last, S^3BB represents the violation probability of the aggregate burstiness B of the whole process. A deterministic arrival curve is a special case of S^3BB, with $\epsilon(b) = 0$, which is why, for a non-zero violation probability $\epsilon(b)$, b is called a *quasi-deterministic* bound on the burstiness.

The first model SBB is the weakest, but also the easiest to handle: bounding the arrivals during a given interval of time can be done for many stochastic models. It was also used for the study of aggregated independent flows with periodic patterns [97, 99, 100, 101, 102, 103]. All the approaches can be summarized as follows: a) defining an event E_s of interest related to some time interval $[s,t)$ and aggregation of the flows; b) combining the events $(E_s)_{s \leq t}$ together to obtain a violation probability of the burstiness or of the backlog bound at time t.

The second model S^{2}BB seems at first more adapted to network calculus analysis, as performance bounds can be directly derived from the formulation. However, the probability bound of S^{2}BB is usually deduced from SBB, which leads to pessimistic bounds for a single server. Nevertheless, this framework may become necessary for more complex cases [173].

In time-sensitive networks, we are interested in the probability that a delay or backlog bound is not violated during some interval (e.g., the network's lifetime), not just one arbitrary point in time, so the two models SBB and S^{2}BB are not adapted, as they do not provide the violation probability of a delay bound during a whole period of interest. In contrast, when using S^{3}BB, we can guarantee, with some probability, that delay and backlog bounds derived by deterministic network calculus are never violated during the network's lifetime, which is why we choose this formulation in our model.

As pointed out in [101, Section 4.4], when arrival processes are stationary and ergodic, S^{3}BB is always trivial and the bounding function $\epsilon(b)$ in (7.6) is either zero or one. This is perhaps why the literature was discouraged from studying S^{3}BB characterizations. However, it has been overlooked that there is interest in some non-ergodic arrival processes, as in our case. Indeed, with our model, phases ϕ_f are drawn randomly but remain the same during the entire period of interest; thus, our arrival processes are not ergodic.

7.3 Homogeneous Case

In this section, we consider the case where flows have the same packet size and same period.

More precisely, we assume

(**H**) There exist $\tau, \ell > 0$ such that $\forall f \in \mathbb{N}_n,\ \ell_f = \ell, \tau_f = \tau$ and $(\phi_f)_{f\in\mathbb{N}_n}$ is a family of independent and identically distributed (iid) uniform random variables (rv) on $[0,\tau)$.

We present two bounds for the aggregate burstiness; the former gives a closed form, unlike the latter, which might be slightly more accurate when the number of flows is small.

Let us first prove a useful result when the period τ is equal to 1; it shows that if the time origin is shifted to the arrival time of the first packet of flow i, the phases of the $n-1$ other flows remain uniformly distributed on $[0,1)$ and mutually independent. For this, we define the function h as $\forall x, y \in [0,1)$,

$$h(x,y) = (x-y)\mathbb{1}_{x\geq y} + (1+x-y)\mathbb{1}_{x<y}. \tag{7.7}$$

Intuitively, if $x = \phi_j$ and $y = \phi_i$, $h(x,y)$ is the time until the arrival of the first packet of flow j, counted from the arrival time of the first packet of flow i.

Lemma 7.1. *Let $U_1, \dots, U_n$ be a sequence of n iid uniform rv on $[0,1)$. Let $i \in \mathbb{N}_n$ and define W_j for $j \in \mathbb{N}_n \setminus \{i\}$ by $W_j = h(U_j, U_i)$. Then, $(W_j)_{j\neq i}$ is a family of $n-1$ iid uniform rv on $[0,1)$.*

Proof. Let us first do a preliminary computation for all $u_i \in [0,1]$ and all bounded measurable function

g_j:

$$\begin{aligned}\mathbb{E}[g_j(h(U_j,u_i))] &= \int_{u_j=0}^{1} g_j(h(u_j,u_i))du_j = \int_{u_j=u_i}^{1} g_j(u_j-u_i)du_j + \int_{u_j=0}^{u_i} g_j(1+u_j-u_i)du_j \\ &= \int_{u_j=0}^{1-u_i} g_j(w_j)dw_j + \int_{u_j=1-u_i}^{1} g_j(w_j)dw_j = \int_{w_j=0}^{1} g_j(w_j)dw_j.\end{aligned}$$

Then, consider a collection of bounded measurable functions $(g_j)_{j\neq i}$: we can compute $\mathbb{E}[\prod_{j\neq i} g_j(W_j)] = \mathbb{E}[\prod_{j\neq i} g_j(h(U_j,U_i))] = \int_{u_i=0}^{1}\mathbb{E}[\prod_{j\neq i} g_j(h(U_j,u_i))]du_i = \int_0^1 \prod_{j\neq i}\mathbb{E}[g_j(h(U_j,u_i))]du_i = \int_0^1\prod_{j\neq i}\mathbb{E}[g_j(V_j)]du_i = \prod_{j\neq i}\mathbb{E}[g_j(V_j)] = \mathbb{E}[\prod_{j\neq i} g_j(V_j)]$, where $(V_j)_{j\neq j}$ is a collection of $n-1$ iid uniformly rv on $[0,1)$. □

The bounds we are to present are based on the order statistics: consider $n-1$ rv $U_1,\dots,U_{n-1}$ and its order statistics is $U_{(1)}\leq\cdots\leq U_{(n-1)}$, defined by sorting $U_1,\dots,U_{n-1}$ in non-decreasing order. It is well-known [174, Equation 1.145] that if (U_i) is an iid family of uniform rv on [0,1], the density function of the joint distribution of $U_{(1)},\dots,U_{(n-1)}$ is

$$f_{U_{(1)},\dots,U_{(n-1)}}(y_1,\dots,y_{n-1}) = (n-1)!\mathbb{1}_{0\leq y_1\leq y_2\leq\dots\leq y_{n-1}\leq 1}. \tag{7.8}$$

The next proposition connects the order statistics of the phases with the aggregate burstiness, and is key for Theorems 7.1 and 7.2.

Proposition 7.1. *Assume model* (**H**). *For all* $0\leq b<n\ell$,

$$\mathbb{P}(B>b)\leq n\mathbb{P}(E), \tag{7.9}$$

with

$$E\overset{def}{=}\bigcup_{k=\lfloor b/\ell\rfloor}^{n-1}\left\{U_{(k)}<\frac{(k+1)-b/\ell}{n}\right\}, \tag{7.10}$$

where $U_{(1)},\dots,U_{(n-1)}$ *is the order statistic of* $n-1$ *iid uniform rv on* $[0,1]$.

Proof. Note that the normalized process $\tilde{A}[s,t)=\frac{1}{\ell}A[\tau s,\tau t)$ follows model (**H**) with $\tau=\ell=1$, and

$$\mathbb{P}(B>b)=\mathbb{P}(\tilde{B}>b/\ell).$$

We then assume in this proof (and that of Theorem 7.1) that $\tau=\ell=1$, and the final result is obtained by replacing b by b/ℓ. One can also remark that the bound is independent of τ.

Let $T_j,\ j\geq 1$ be the arrival time of the j-th packet in the aggregate. With probability 1, T_j is strictly increasing as we assume all phases are different. First, for all $i\leq j$, for all $(t_i,t_j)\in(T_{i-1},T_i]\times(T_j,T_{j+1}]\overset{def}{=}C_{i,j}$, $A[t_i,t_j)=j-i+1$, and $H_{i,j}\overset{def}{=}\sup_{t_i,t_j\in C_{i,j}}A[t_i,t_j)-n(t_i-t_j)=j-i+1-n(T_j-T_i)$. Then, we can rewrite the aggregate burstiness as

$$B=\sup_{1\leq i\leq j}\sup_{t_i,t_j\in C_{i,j}}A[t_i,t_j)-n(t_i-t_j)=\sup_{1\leq i\leq j}H_{i,j}. \tag{7.11}$$

As our model is the aggregation of n flows of period 1, $T_{j+n}=T_j+1$ for $j\geq 1$, and $H_{i,j+n}=j+n-i+1-n(T_j-1-T_i)=H_{i,j}$ for all $j\geq i$. Similarly, $H_{i+n,j}=H_{i,j}$ for all $j\geq i+n$. Combine this with (7.11)

and obtain

$$B = \max_{j\geq 1} \max_{i\in\mathbb{N}_j} H_{i,j} = \max_{i\in\mathbb{N}_n} \underbrace{\max_{i\leq j\leq n-1} H_{i,j}}_{B_i}. \tag{7.12}$$

We now prove that $\forall\, i \in \mathbb{N}_n$, $\mathbb{P}(B_i > b) = \mathbb{P}(E)$.

Observe that for all $j \geq i$, we have the equality of events $\{H_{i,j} > b\} = \{T_j - T_i < (j-i+1-b)/n\}$, so for all $i \in \mathbb{N}_n$, $\{B_i > b\} = \bigcup_{j=i}^{i+n-1} \left\{T_j - T_i < (j-i+1-b)/n\right\}$.

We can also notice that the sequence $(T_j - T_i)_{j=i+1}^{n+i-1}$ is the ordered sequence of phases starting from time origin T_i. Conditionally to $T_i = \phi_f$, or equivalently $\phi_{(i)} = f$, $(T_j - T_i)_{j=i+1}^{n+i-1}$ is the order statistics of $(\phi_j - \phi_f)_{j\neq f}$, which is, from Lemma 7.1, iid and uniformly distributed on $[0,1)$. If follows that

$$\mathbb{P}(B_i > b \mid \phi_{(i)} = f) = \mathbb{P}(\cup_{k=1}^{n-1}\{U_{(k)} < \frac{k-b}{n}\}) = \mathbb{P}(\cup_{k=\lfloor b\rfloor}^{n-1}\{U_{(k)} < \frac{k-b}{n}\}) = \mathbb{P}(E),$$

since $U_{(k)} \geq 0$. Then, using the law of total probabilities, $\mathbb{P}(B_i > b) = \sum_{f=1}^{n} \mathbb{P}(B_i > b \mid \phi_{(i)} = f)\mathbb{P}(\phi_{(i)} = f) = \mathbb{P}(E)$.

Lastly, we conclude by using the union bound: $\mathbb{P}(B > b) = \mathbb{P}(\cup_{i=1}^{n} B_i > b) \leq \sum_{i=1}^{n} \mathbb{P}(B_i > b) = n\mathbb{P}(E)$. □

We now present the first bound on the tail probability of the aggregate burstiness B.

Theorem 7.1 (Homogeneous case, DKW bound)**.** *Assume model* (**H**) *with* $n > 1$. *For all* $b < n\ell$, *a bound on the tail probability of the aggregate burstiness* B *is given by*

$$\mathbb{P}(B > b) \leq n \exp\left(-2(n-1)\left(\frac{\lfloor b/\ell\rfloor}{n-1} - \frac{1}{n}\right)^2\right) \overset{def}{=} \varepsilon^{dkw}(n,\ell,b). \tag{7.13}$$

Proof. Let us assume that $\tau = \ell = 1$ in the proof, as in the proof of Proposition 7.1. Observe that when $\lfloor b\rfloor < 1 - \frac{1}{n} + \sqrt{\frac{(n-1)\log 2}{2}}$, we have $\varepsilon^{\text{dkw}}(n,1,b) \geq \frac{n}{2}$, hence (7.13) holds. Therefore we now proceed to prove (7.13) when $\lfloor b\rfloor \geq 1 - \frac{1}{n} + \sqrt{\frac{(n-1)\log 2}{2}}$.

Step 1: Consider $n-1$ iid, rv $U_1, \ldots, U_{n-1}$ and its order statistics is $U_{(1)} \leq \cdots \leq U_{(n-1)}$, defined by sorting $U_1, \ldots, U_{n-1}$ in non-decreasing order. For $\varepsilon > 0$, define $E'(\varepsilon)$ by

$$E'(\varepsilon) \overset{def}{=} \bigcup_{k=1}^{n-1} \left\{U_{(k)} < \frac{k}{n-1} - \varepsilon\right\}. \tag{7.14}$$

We now show that if $\varepsilon \geq \sqrt{\frac{\log 2}{2(n-1)}}$,

$$\mathbb{P}\left(E'(\varepsilon)\right) \leq e^{-2(n-1)\varepsilon^2}. \tag{7.15}$$

Let F_{n-1} be the (random) empirical cumulative distribution function of $U_1, \ldots, U_{n-1}$, defined $\forall\, x \in [0,1]$ by

$$F_{n-1}(x) = \frac{1}{n-1} \sum_{i=1}^{n-1} \mathbb{1}_{U_{(i)}\leq x}. \tag{7.16}$$

The Dvoretzky–Kiefer–Wolfowitz inequality [170] states that if $\varepsilon \geq \sqrt{\frac{\log 2}{2(n-1)}}$, then

$$\mathbb{P}\Big(\sup_{x\in[0,\,1]} (F_{n-1}(x) - x) > \varepsilon \Big) \leq e^{-2(n-1)\varepsilon^2}. \tag{7.17}$$

We can apply this to find the bound of interest. First, we prove that

$$\sup_{x\in[0,\,1]} (F_{n-1}(x) - x) > \varepsilon \Leftrightarrow \exists k \in \mathbb{N}_{n-1},\ U_{(k)} < \frac{k}{n-1} - \varepsilon. \tag{7.18}$$

Proof of $\Leftarrow$: First, observe that $F_{n-1}(U_{(k)}) = k/(n-1)$, so if $\frac{k}{n-1} - U_{(k)} > \varepsilon$ for some k, then $F_{n-1}\left(U_{(k)}\right) - U_{(k)} > \varepsilon$, and the left-hand side holds.

Proof of $\Rightarrow$: Set $U_{(0)} = 0$ and $U_{(n)} = 1$. Observe that for all $k \in \{0,\ldots,n-1\}$, and all $U_{(k)} \leq x < U_{(k+1)}$, $F_{n-1}(x) = F_{n-1}\left(U_{(k)}\right) = \frac{k}{n-1}$. Hence, $F_{n-1}(x) - x = \frac{k}{n-1} - x$ is decreasing on each segment $[U_{(k)}, U_{(k+1)})$. Then, the supremum in the left-hand side of (7.18) is obtained for some $x = U_{(k)}$, i.e., $\sup_{x\in[0,\,1]}(F_{n-1}(x) - x) = \sup_{k\in\{0,\ldots,n-1\}}(F_{n-1}(U_{(k)}) - U_{(k)}) = \sup_{k\in\{0,\ldots,n-1\}}(\frac{k}{n-1} - U_{(k)})$, which implies the right-hand side ($F_{n-1}(0) - 0 = 0 < \varepsilon$).

This proves (7.18), and Step 1 is concluded by combining it with (7.17).

Step 2: We now proceed to show that if

$$\varepsilon = \frac{\lfloor b \rfloor}{n-1} - \frac{1}{n}, \tag{7.19}$$

then, $E \subseteq E'(\varepsilon)$, where event E is defined in Proposition 7.1.

It is enough to show that for all $k \in \{\lfloor b \rfloor,\ldots,n-1\}$, $\frac{k+1-b}{n} \leq \frac{k-\lfloor b \rfloor}{n-1} + \frac{1}{n}$, which can be deduced from the following implications:

$$\frac{k+1-b}{n} \leq \frac{k-\lfloor b \rfloor}{n-1} + \frac{1}{n} \Leftrightarrow \frac{k-b}{n} \leq \frac{k-\lfloor b \rfloor}{n-1} \Leftarrow \frac{k-\lfloor b \rfloor}{n} \leq \frac{k-\lfloor b \rfloor}{n-1} \Leftrightarrow \frac{1}{n} \leq \frac{1}{n-1}.$$

Step 3: By Step 2, we have $\mathbb{P}(E) \leq \mathbb{P}(E'(\varepsilon))$. Also, observe that $\lfloor \frac{b}{l} \rfloor \geq 1 - \frac{1}{n} + \sqrt{\frac{(n-1)\log 2}{2}}$ implies $\varepsilon \geq \sqrt{\frac{\log 2}{2(n-1)}}$. Thus, combine it with Step 1 to obtain

$$\mathbb{P}(E) \leq \mathbb{P}(E'(\varepsilon)) \leq \exp\Big(-2(n-1)\Big(\frac{\lfloor b \rfloor}{n-1} - \frac{1}{n}\Big)^2\Big). \tag{7.20}$$

Combine (7.20) with Proposition 7.1 to conclude the theorem.

□

Note that the bound of Theorem 7.1 is only less than one and is non-trivial when $\lfloor \frac{b}{l} \rfloor \geq 1 - \frac{1}{n} + \sqrt{\frac{(n-1)\log 2}{2}}$.

The following corollary provides a closed-form formulation for the minimum value for the aggregate burstiness with a violation probability of at most ε. It is obtained by setting the right-hand side of (7.13) in Theorem 7.1 to ε.

Corollary 7.1 (Quasi-deterministic burstiness bound)**.** *Assume model* (**H**) *with* $n > 1$. *Consider some*

$0 < \varepsilon < 1$, *and define*

$$b(n,\ell,\varepsilon) \stackrel{def}{=} \ell\left\lceil 1 - \frac{1}{n} + \sqrt{\frac{(n-1)(\log\, n - \log\, \varepsilon)}{2}}\right\rceil. \tag{7.21}$$

Then, $b(n,\ell,\varepsilon)$ is a quasi-deterministic burstiness bound for the aggregate with the violation probability of at most ε, i.e., $\mathbb{P}(B > b(n,\ell,\varepsilon)) \le \varepsilon$.

Observe that $b(n,\ell,\varepsilon)$ grows in $\sqrt{n\log n}$ as opposed to the deterministic bound ($\ell^{\text{tot}} = n\ell$) that grows in linearly (see Fig. 7.5.1b).

Proposition 7.1 introduces the event E such that an upper bound of $\mathbb{P}(E)$ is used to derive an upper bound on the tail probability of the aggregate burstiness. Theorem 7.1 is derived from the DKW upper bound of $\mathbb{P}(E)$, which is tight when the number of flows n is large. In Theorem 7.2, we compute the exact value of $\mathbb{P}(E)$; thus, it provides a slightly better bound when the number of flows is small but at the expense of not having a closed-form expression.

Theorem 7.2 (Refinement of Theorem 7.1 for small groups)**.** *Assume model* (**H**) *with $n > 1$. For all $b \ge 0$. Then, a bound on the tail probability of the aggregate burstiness B is*

$$\mathbb{P}(B > b) \le n(1 - p(n,\ell,b)) \stackrel{def}{=} \varepsilon^{thm2}(n,\ell,b), \tag{7.22}$$

with

$$p(n,\ell,b) = (n-1)!\int_{y_{n-1}=u_{n-1}}^{1}\int_{y_{n-2}=u_{n-2}}^{y_{n-1}}\cdots\int_{y_i=u_i}^{y_{i+1}}\cdots\int_{y_1=u_1}^{y_2} 1\, dy_1 \ldots dy_{n-1}, \tag{7.23}$$

and $u_k = \frac{[(k+1)-b/\ell]^+}{n}$, for all $k \in \mathbb{N}_{n-1}$ and $[x]^+ = \max(0,x)$.

Note that the computation of the bound of Theorem 7.2 requires computing $p(n,\ell,b)$ in (7.23), which is a series of polynomial integrations, and finding a general closed-form formula might be challenging. However, computing the bound can be done iteratively as in Algorithm 7.1: The integrals are computed from the inner sign to the outer (incorporation factor i from the factorial in the i-th integral). Polynoms are computed at each step and variable q_j^m represents the coefficient of degree j of the m-th integral. Note that we always have $q_m^m = 1$, so the monomial of degree $n-1$ cancels in (7.22).

All computations involve exact representations of the integrals (no numerical integration) and use exact arithmetic with rational numbers; therefore, the results are exact with infinite precision.

Algorithm 7.1: Computation of $\varepsilon^{\text{thm2}}(n,\ell,b)$ from Theorem 7.2

Inputs : number of flows n, a burst b, and a packet size ℓ.
Output : $\varepsilon^{\text{thm2}}(n,\ell,b)$ such that $\mathbb{P}(B > b) \le \varepsilon^{\text{thm2}}(n,\ell,b)$.

$m \leftarrow \lfloor b/\ell \rfloor - 1$;
$\left(q_0^m, q_1^m, \ldots, q_m^m\right) \leftarrow (0,0,\ldots,0,1)$;
for $m \leftarrow \lfloor b/\ell \rfloor$ **to** $n-1$ **do**
  $u_m \leftarrow (m+1-b/\ell)/n$;
  $q_0^m \leftarrow -\sum_{j=0}^{m-1} \frac{m q_j^{m-1}}{j+1} u_m^{j+1}$;
  for $i \leftarrow 1$ **to** m **do** $q_i^m \leftarrow \frac{m q_{i-1}^{m-1}}{i}$;
return $n\sum_{i=0}^{n-2} q_i^{n-1}$

Proof. Let $\bar{E}$ be the complementary event of E defined in Proposition 7.1.

$$\bar{E} = \bigcap_{k=1}^{n-1} \Big\{ U_{(k)} \geq \frac{[k+1-b/\ell]^+}{n} \Big\}. \tag{7.24}$$

Let $f_{U_{(1)},\ldots,U_{(n-1)}}$ be the density function of the joint distribution of $U_{(1)},\ldots,U_{(n-1)}$, given in (7.8). Then

$$\mathbb{P}\left(\bar{E}\right) = \int_{y_{n-1}=u_{n-1}}^{1} \cdots \int_{y_i=u_i}^{1} \cdots \int_{y_1=u_1}^{1} f_{U_{(1)},\ldots,U_{(n-1)}}\left(y_1,\ldots,y_{n-1}\right) dy_1 \ldots dy_{n-1} \tag{7.25}$$

$$= \int_{y_{n-1}=u_{n-1}}^{1} \cdots \int_{y_i=u_i}^{1} \cdots \int_{y_1=u_1}^{1} (n-1)! \mathbb{1}_{0\leq y_1 \leq y_2 \leq \ldots \leq y_{n-1} \leq 1} \, dy_1 \ldots dy_{n-1} \tag{7.26}$$

$$= (n-1)! \int_{y_{n-1}=u_{n-1}}^{1} \cdots \int_{y_i=u_i}^{y_{i+1}} \cdots \int_{y_1=u_1}^{y_2} 1 \, dy_1 \ldots dy_{n-1} = p(n,\ell,b). \tag{7.27}$$

Combine it with $\mathbb{P}\left(\bar{E}\right) = 1 - \mathbb{P}\left(E\right)$ and Proposition 7.1 to conclude the theorem. □

Note that since Theorem 7.2 computes the exact probability of event E, we have $\varepsilon^{\text{dkw}}(n,\ell,b) \geq \varepsilon^{\text{thm2}}(n,\ell,b)$.

7.4 Heterogeneous Case

In this section, we consider the case where flows have different periods and packet sizes. We present burstiness bounds in two different settings: First, when flows can be grouped into homogeneous flows; second, when all packets have the same period but with different packet sizes.

Let us first focus on the model where flows are grouped according to their characteristics:

(G) There exists a partition $I_1,\ldots,I_g$ of $\mathbb{N}_n$ such that I_i is a group of n_i flows satisfying model (**H**) with packet size ℓ_i and period τ_i. All phases are mutually independent.

Proposition 7.2 (Convolution Bound)**.** *Let $X_1, X_2, \ldots, X_g$ be $g \geq 1$ mutually independent rv on $\mathbb{N}$. Assume that for all $i \in \mathbb{N}_n$, Ψ_i is wide-sense increasing and is a lower bound on the CDF of X_i, namely, $\forall b \in \mathbb{N}$, $\mathbb{P}(X_i \leq b) \geq \Psi_i(b)$. Define ψ_i by $\psi_i(0) = \Psi_i(0)$ and $\psi_i(b) = \Psi_i(b) - \Psi_i(b-1)$ for $b \in \mathbb{N} \setminus \{0\}$.*

Then, a lower bound on the CDF of $\sum_{i=1}^{g} X_i$ is given by: $\forall b \in \mathbb{N}$,

$$\mathbb{P}\Big(\sum_{i=1}^{g} X_i \leq b\Big) \geq \left(\psi_1 * \psi_2 * \cdots * \psi_{g-1} * \Psi_g\right)(b), \tag{7.28}$$

where, the symbol $*$ denotes the discrete convolution, defined for arbitrary functions $f_1, f_2 : \mathbb{N} \to \mathbb{R}$ by

$$\forall b \in \mathbb{N}, \quad (f_1 * f_2)(b) = \sum_{j=0}^{b} f_1(j) f_2(b-j). \tag{7.29}$$

Proof. We prove it by induction on g.

Base Case $g = 1$**:** There is nothing to prove: for all $b \in \mathbb{N}$, $\mathbb{P}(X_1 \leq b) \geq \Psi_1(b)$.

Induction Case: We now assume that Equation (7.28) holds for g variables, and we show that it also holds for $g+1$ variables.

We can apply Equation (7.28) to variables $X_2, X_3, \ldots, X_{g+1}$, and let us denote $Y = X_2 + \cdots + X_{g+1}$ and $\Psi = \psi_2 * \cdots * \psi_g * \Psi_{g+1}$. We need to show that for all $b \in \mathbb{N}$,

$$\mathbb{P}(X_1 + Y \leq b) \geq (\psi_1 * \Psi)(b). \tag{7.30}$$

Let $F(b) = \mathbb{P}(Y \leq b)$ and observe that $\mathbb{P}(Y = 0) = F(0)$ and $\mathbb{P}(Y = b) = F(b) - F(b-1)$ for $b \in \mathbb{N} \setminus \{0\}$. Then, since X_1 and Y are independent,

$$\mathbb{P}(X_1 + Y \leq b) = \sum_{j=0}^{b} \mathbb{P}(X_1 + j \leq b | Y = j)\mathbb{P}(Y = j) = \sum_{j=0}^{b} \mathbb{P}(X_1 + j \leq b)\mathbb{P}(Y = j) \tag{7.31}$$

$$\geq \sum_{j=0}^{b} \Psi_1(b-j)\mathbb{P}(Y = j) \tag{7.32}$$

$$\geq \Psi_1(b)F(0) + \sum_{j=1}^{b} \Psi_1(b-j)(F(j) - F(j-1)). \tag{7.33}$$

We now use Abel's summation by parts in (7.33) and obtain

$$\mathbb{P}(X_1 + Y \leq b) \geq \Psi_1(b)F(0) + \sum_{j=1}^{b} \Psi_1(b-j)F(j) - \sum_{j=1}^{b} \Psi_1(b-j)F(j-1) \tag{7.34}$$

$$= \Psi_1(b)F(0) + \sum_{j=1}^{b} \Psi_1(b-j)F(j) - \sum_{j=0}^{b-1} \Psi_1(b-j-1)F(j) \tag{7.35}$$

$$= \sum_{j=0}^{b} \Psi_1(b-j)F(j) - \sum_{j=0}^{b-1} \Psi_1(b-j-1)F(j) \tag{7.36}$$

$$= \Psi_1(0)F(b) + \sum_{j=0}^{b-1} (\Psi_1(b-j) - \Psi_1(b-j-1))F(j) \tag{7.37}$$

$$= \psi_1(0)F(b) + \sum_{j=0}^{b-1} \psi_1(b-j)F(j) = \sum_{j=0}^{b} \psi_1(b-j)F(j) \tag{7.38}$$

$$\geq \sum_{j=0}^{b} \psi_1(b-j)\Psi(j) = (\psi_1 * \Psi)(b). \tag{7.39}$$

We can conclude by using the associativity of the discrete convolution: $\psi_1 * \Psi = \psi_1 * \cdots * \psi_g * \Psi_{g+1}$. □

Remarks. 1. Note that $(\psi_1 * \Psi_2)(b) = \sum_{i+j \leq b} \psi_1(i) + \psi_2(j) = (\psi_2 * \Psi_1)(b)$, so the convolution bound is independent of the order of $X_1, \ldots, X_g$.

2. An alternative to Proposition 7.2 is to use then union bound rather than the convolution bound: for all $(b_1, \ldots, b_g) \in \mathbb{N}^g$ such that $\sum_{i=1}^{g} b_i = b$, we have $\left\{\sum_{i=1}^{g} X_i > b\right\} \subseteq \bigcup_{i=1}^{g} \{X_i > b_i\}$, so $\mathbb{P}(X > b) \leq \sum_{i=1}^{g} \mathbb{P}(X_i > b_i) \leq \sum_{i=1}^{g} (1 - \Psi_i(b_i))$. We can choose $(b_i)_{i=1}^{g}$ so as to minimize this latter term, and take the complement to obtain

$$\mathbb{P}(\sum_{i=1}^{g} X_i \leq b) \geq 1 - \min_{b_1 + \cdots + b_g = b} \sum_{i=1}^{g} (1 - \Psi_i(b_i)). \tag{7.40}$$

This bound is also valid when rvs X_i are not independent, but it can be shown that the convolution bound always dominates the union bound. In our numerical evaluations, we find that the convolution

bound provides significantly better results than the union bound.

Theorem 7.3 (Flows with different periods and different packet-sizes)**.** *Assume model* (**G**)*. Let ε_i be a wide-sense decreasing function that bounds the tail probability of aggregate burstiness B_i of each group $i \in \mathbb{N}_g$: for all $b \in \mathbb{N}$, $\mathbb{P}(B_i > b) \le \varepsilon_i(b)$ for all $b \in \mathbb{N}$. Define $\Psi_i(b) = 1 - \varepsilon_i(b)$ for $b \in \mathbb{N}$ and define ψ_i by $\psi_i(0) = \Psi_i(0)$ and $\psi_i(b) = \varepsilon_i(b-1) - \varepsilon_i(b)$ for $b \in \mathbb{N} \setminus \{0\}$.*

Then, a bound on the tail probability of the aggregate burstiness of all flows B is given by $\forall\, b \in \mathbb{N}_{\ell^{tot}}$,

$$\mathbb{P}(B > b) \le 1 - \big(\psi_1 * \psi_2 * \cdots * \psi_{g-1} * \Psi_g\big)(b), \tag{7.41}$$

Proof. For all group $i \in \mathbb{N}_g$, let $A^i[s,t)$ be the aggregate of flows of group i during the interval $[s,t)$, r^i, its aggregate arrival rate, and B_i its aggregate burstiness. Observe that for all $s \le t$, $A(s,t] = \sum_{i=1}^g A^i[s,t)$ and $r^{\text{tot}} = \sum_{i=1}^g r^i$. We then obtain

$$B = \sup_{0\le s\le t} \{A(s,t] - r^{\text{tot}}(t-s)\} = \sup_{0\le s\le t} \Big\{ \sum_{i=1}^g \big(A_i(s,t] - r_i(t-s)\big)\Big\} \tag{7.42}$$

$$\le \sum_{i=1}^g \sup_{0\le s\le t} \{A_i(s,t] - r_i(t-s)\} = \sum_{i=1}^g B_i \le \sum_{i=1}^g \lceil B_i \rceil. \tag{7.43}$$

Hence, it follows that $\mathbb{P}(B \le b) \ge \mathbb{P}(\sum_{i=1}^g \lceil B_i \rceil \le b), \quad b \in \mathbb{N}$.

We now apply Proposition 7.2 with $X_i = \lceil B_i \rceil$ and Ψ_i as defined in the theorem: it suffices to observe that $(\lceil B_i \rceil)_{i\in\mathbb{N}_g}$ are mutually independent rv on $\mathbb{N}$; as ε_i is wide-sense decreasing, Ψ_i is wide-sense increasing; Hence, by Proposition 7.2, we obtain that for all $b \in \mathbb{N}$, $\mathbb{P}(\sum_{i=1}^g \lceil B_i \rceil \le b) \ge (\psi_1 * \psi_2 * \ldots * \psi_{g-1} * \Psi_g)(b)$, which concludes the proof.

□

We now turn to our second heterogeneous model: when all flows have the same period but different packet sizes.

(**P**) There exists $\tau > 0$ such that $\forall\, f \in \mathbb{N}_n$, $\tau_f = \tau$; $\ell_1 \ge \ell_2 \ge \cdots \ge \ell_n > 0$ and $(\phi_f)_{f\in\mathbb{N}_n}$ is a family of iid uniform rv on $[0,\tau)$.

Theorem 7.4 (Flows with the same period but different packet sizes)**.** *Assume model* (**P**)*. For all $0 \le b < \ell^{tot}$, set $\eta \stackrel{def}{=} \min\Big\{ \frac{k}{n-1} - \frac{\sum_{j=1}^{k+1} \ell_j}{\ell^{tot}},\ k \in \mathbb{N}_{n-1}, \sum_{j=1}^{k+1} \ell_j > b \Big\}$. Then*

1. *A bound on the tail probability of the aggregate burstiness of all flows B is*

$$\mathbb{P}(B > b) \le n \exp\Big(-2(n-1)\big(\eta + \frac{b}{\ell^{tot}}\big)^2 \Big). \tag{7.44}$$

2. *For all $\varepsilon \in (0,1)$, for all $n \ge 2$, the violation probability of at most ε, i.e., $\mathbb{P}(B > b(n,\ell_1,\ldots,\ell_n,\varepsilon)) \le \varepsilon$ with*

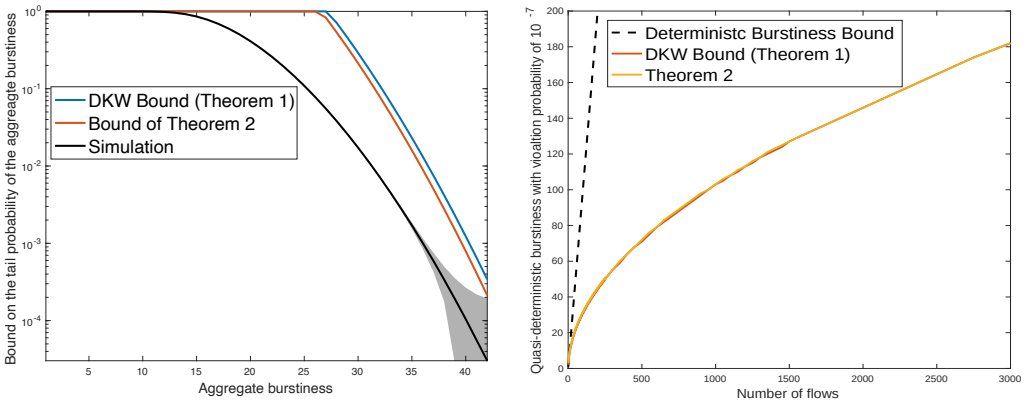
$$b(n,\ell_1,\ldots,\ell_n,\varepsilon) \stackrel{def}{=} \ell^{tot} \Big\lceil \sqrt{\frac{\log n - \log \varepsilon}{2(n-1)}} - \eta \Big\rceil. \tag{7.45}$$

3. *A bound on the tail probability of the aggregate burstiness of all groups B is given by* $\mathbb{P}(B > b) \leq n(1-\bar{p}(n,\ell_1,\ldots,\ell_n,b))$, *where* $\bar{p}(n,\ell_1,\ldots,\ell_n,b)$ *is computed as in Equation* (7.23), *where for all* $k \in \mathbb{N}_{n-1}$, $u_k = \frac{[\sum_{j=1}^{k+1}\ell_j - b]^+}{\ell^{tot}}$.

When all flows have the same packet-sizes ℓ, this is model (**H**) and the bounds provided are exactly the same as in Section 7.3. Algorithm 7.1 can also be used to compute the bound of item 3 if a) line 1 is replaced by $m \leftarrow \max\{k \geq 0 \mid \sum_{j=1}^{k+1}\ell_j \leq b\}$ and b) the values of u_m are adapted in line 4.

Proof. The proof is done by adapting Proposition 7.1. Then the proofs of each item follow exactly the steps of Theorems 7.1, Corollary 7.1 and Theorem 7.2. The key difference in Proposition 7.1 is the computation of $H_{i,j}$: $H_{i,j} \leq \sum_{k=1}^{j-i+1}\ell_k - \ell^{\text{tot}}(T_j - T_i)$: we bound this value as if the packets arrived in this arrival where the $j-i+1$ longest ones. □

7.5 Numerical Evaluation



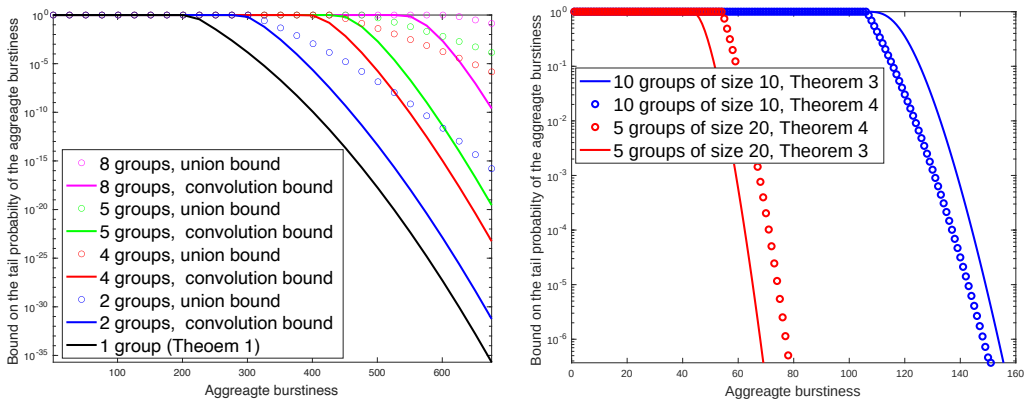
(a) (b)

Figure 7.5.1: (a): Bound on the tail probability of the aggregate burstiness obtained by Theorems 7.1, 7.2, and simulations. (b): The obtained quasi-deterministic burstiness with violation probability of 10^{-7} by Corollary 7.1 and Theorem 7.2, as the number of flows grows; the deterministic bound (dashed plot) grows linearly with the number of flows.

In this section, we numerically illustrate our bounds in Fig. 7.5.1 and Fig. 7.5.2.

7.5.1 Homogeneous Case

In Fig. 7.5.1 (a), we consider 250 flows with the same packet size (with respect to a unit, is assumed to be 1) and the same period. We then compute bounds on the tail probability of their aggregate burstiness using Theorems 7.1 and 7.2. We also compute the bound using simulations: For each flow, we independently pick a phase uniformly at random, and we then compute the aggregate burstiness as in (7.1); we repeat this 10^8 times. We then compute bounds on the tail probability of their aggregate burstiness and its 99% Kolmogorov–Smirnov confidence band. The bound of Theorem 7.2 is slightly better than that of Theorem 7.1. Also, compared to simulations, our bounds are fairly tight.



(a) (b)

Figure 7.5.2: (a): Comparison of the convolution bound of Theorem 7.3 to the union bound when combining bound obtained for homogeneous sets of flows. (b): Slight improvement of Theorem 7.4 compared to Theorem 7.3 when the number of flows per packet-size is small.

In Fig. 7.5.1 (b), we consider $n \in \{2,\dots,3000\}$ flows with the packet size 1 and same period. We then compute a quasi-deterministic burstiness bound with violation probability of 10^{-7} once using Corollary 7.1 and once using Theorem 7.2; they are almost equal and as n grows are exactly equal, as Theorem 7.1 is as tight as Theorem 7.2 for large n. Also, our quasi-deterministic burstiness bound is considerably less than the deterministic one (i.e., n) and grows in $\sqrt{n \log n}$.

7.5.2 Heterogeneous Case

To assess the efficiency of the bound in the heterogeneous case, we consider in Fig. 7.5.2 (a) 10000 homogeneous flows with period and packet length 1, and divide them into g groups of $10000/g$ flows, for $g \in \{1,2,4,5,8\}$. We compute a bound for each group by Theorem 7.1, and combine them once with the convolution bound of Theorem 7.3 and once by the union bound (as explained after Proposition 7.2). Our convolution bound is significantly better than the union bound, and the differences increase fast with the number of sets.

In Fig. 7.5.2 (b), we consider 10 (resp. 5) homogeneous groups of 10 (resp. 20) flows, flows of each set $g \in \mathbb{N}_{10}$ (resp. $g \in \mathbb{N}_5$), have a packet-size equal to g, and all flows have the same period. We then compute the bound on the tail probability of the aggregate burstiness once with Theorem 7.3 and once with Theorem 7.4. When groups are small (here of 10 flows), Theorem 7.4 provides better bounds than Theorem 7.3, but when groups are larger (here of 20 flows), Theorem 7.3 dominates Theorem 7.4.

7.6 Conclusion

In this chapter, we provided quasi-deterministic bounds on the aggregate burstiness for independent, periodic flows. When a small violation tolerance, is allowed, the bounds are considerably better compared to the deterministic bounds. We obtained a closed-form expression for the homogeneous case, and for the heterogeneous case, we combined bounds obtained for homogeneous sets using the convolution bounding technique. We on purpose limited our study to the burstiness. Quasi-

deterministic delay and backlog bounds can be obtained by applying any method from deterministic network calculus, and combining it with our results.

7.7 Notation

Table 7.7.1: Notation List, Specific to Chapter 7

f	A (periodic) flow
n	Number of flows
ℓ_f	Packet length of flow f
τ_f	Period of flow f
ϕ_f	Phase of flow f
r_f	Rate of flow f, equal to $\frac{\ell_f}{\tau_f}$
B	Aggregate burstiness
$\bar{B}(t)$	Aggregate burstiness when observing the aggregate traffic up to time t
ℓ^{tot}	Aggregate packet length
r^{tot}	Aggregate rate
$A[s,t)$	Number of bits observed in time interval $[s,t)$ for the aggregate traffic
λ_c	Rate function with $\lambda_c(t) = ct$
$\beta_{R,L}$	Rate-latency function with $\beta_{c,L}(t) = \max(0, c(t-L))$
$\mathbb{N}$	$\{1,2,3,\ldots\}$
$\mathbb{N}_n$	$\{1,2,3,\ldots,n\}$
$[x]^+$	$[x]^+ = \max(0,x)$
$*$	discrete convolution $\forall b \in \mathbb{N}, \quad (f_1 * f_2)(b) = \sum_{j=0}^{b} f_1(j) f_2(b-j)$

Conclusion Part IV

8 Conclusion and Future Works

In the first part of this thesis, we have provided the best-known, proven worst-case delay analysis of time-sensitive networks of generic topology with round-robin schedulers, which dramatically dominate all previous works:

- In Chapter 3, we have provided a residual strict service curve for IWRR, a variant of WRR with the same long-term rate and the same complexity but with smoother service. We show that the IWRR strict service curve is the best possible one under general assumptions, thanks to the method of the pseudo-inverse. For classes with packets of constant size, we have shown that the delay bounds derived from it are worst-case. We have proved that IWRR worst-case delay is not greater than WRR and shown on experiments that the gain is significant in practice, which speaks in favor of using IWRR as a replacement to WRR.

  Future Works: Our IWRR strict service curves are obtained under the assumption that all packets of the interfering traffic are of the maximum packet size and all packets of the class of interest are of the minimum packet size; this can be improved with supplementary hypotheses on classes and considering packet size distribution, with "packet curves" [175]. Also, as explained in Section 3.2, two versions for IWRR are presented in [84], which we analyzed one of them. The other version, called List-Based IWRR (LBIWRR), obtains even a smoother service than IWRR, but the analysis might be more complicated.

- In Chapter 4, the method of the pseudo-inverse enabled us to perform a detailed analysis of DRR and to obtain strict service curves that significantly improve the previous results. Our results use the network calculus approach and are mathematically proven, unlike some previous delay bounds that we have proved to be incorrect. Our method assumes that the aggregate service provided to the DRR subsystem is modeled with a strict service curve. Therefore it can be recursively applied to hierarchical DRR schedulers as found, for instance, with class-based queuing.

  Future Works: Our strict service curves in the degraded operational mode (i.e., when there is no assumption on the arrival curves of interfering classes) are the best possible ones; however, for the non-degraded operational mode (i.e., when some arrival curves can be assumed for interfering classes), our strict service curves are the best possible ones so far, but the jury is still out on them. Also, the same method can be applied to IWRR and results in strict service curves for IWRR that account for the arrival curves of the interfering traffic.

- In Chapter 5, we solved the problem of how to combine DRR strict service curves and the

network analysis of PLP in order to obtain worst-case delay bounds in time-sensitive networks. Our method is guaranteed to find delay bounds that are at least as good as the state-of-the-art, and we found very significant improvements for the industrial network under study. It is based on a generic shared memory execution model, implementations of which can differ by the scheduling of the individual operations in the refinement phase. We proved that all implementations produce the same bounds. We proposed two concrete implementations and found that the latter performs faster.

Future Works: It will be interesting to study other concrete implementations that aim at reducing computing time, and to have a publicly available implementation. Also, we enabled PLP to support non-convex service curves; the same method can be applied for arrival curves and to enable PLP to support non-affine arrival curves, for example stair functions.

In the second part of this thesis, we have provided the best-known, proven worst-case delay analysis of time-sensitive networks of generic topology with many periodic flows that, while remaining computationally feasible, dramatically reduce the pessimism of existing works:

- TFA quickly becomes intractable when there is a large number of periodic sources and UPP curves. This problem is frequently observed in time-sensitive and real-time networks. In Chapter 6, we provided a practical solution, FH-TFA, that obtains delay bounds that are formally proven, and can handle a large number of periodic sources with different periods.

  Future Works: It will be interesting to explore other versions of TFA that do not require cuts, such as AsyncTFA and AltTFA [64], and to also make them practical and applicable in the presence of a large number of periodic sources and UPP curves. This is of interest as FH-TFA is currently based on GFP-TFA, which is a sequential algorithm, but other versions of TFA can benefit from parallel computations, thus might further reduce the run times.

- In Chapter 7, we provided quasi-deterministic bounds on the aggregate burstiness for independent, periodic flows. When a small violation tolerance, is allowed, the bounds are considerably better compared to the deterministic bounds. We obtained a closed-form expression for the homogeneous case, and for the heterogeneous case, we combined bounds obtained for homogeneous sets using the convolution bounding technique.

  Future Works: Our method uses the affine arrival curves, and it will be interesting to obtain quasi-deterministic arrival curve constraints but with UPP curves.

Lastly, as a side contribution, in Appendix A, we presented Saihu, a Python interface capable of executing multiple network analysis tools easily. It enables users to define a network and retrieve the analysis results for multiple tools. Its unified input and output layers enable the automation of all the tedious re-interpretation of a network for each tool. Such design not only simplifies many mechanical procedures previously hindering network researchers' works but also helps publicize these useful tools to more potential users due to its simplicity.

Future Works: Saihu is modular and hopefully will also be extended to other analysis tools.

Appendix Part V

A Saihu : A Common Interface of Worst-Case Delay Analysis Tools for Time-Sensitive Networks

In time-sensitive networks, where bounds reside,
Delays of utmost importance, a need to confide.
Tools aplenty, methods diverse,
Valid bounds they offer, yet which is to immerse?

IEEE-TSN and IETF-Detnet, the realms we tread,
Worst-case delays, the challenge ahead.
Different tools, different paths they pave,
Uncertainty lingers, which bound to save?

Implementing multiple codes, a cumbersome chore,
Syntax variations, errors galore.
Impracticality looms, a burden to bear,
A solution we seek, a simpler affair.

Saihu emerges, a Python interface in sight,
xTFA, DiscoDNC, Panco unite.
Three tools combined, six methods entwined,
A unified interface, simplicity defined.

Networks defined in a single file,
Tools executed, results compiled with style.
Formatted reports, effortlessly exported,
Time-sensitive networks, easily supported.

With Saihu as our guide, the burden dissolves,
Accessible to all, as it evolves.
Simplified execution, a user's delight,
Time-sensitive networks, now within our sight.

Created with ChatGPT, free research preview (version May 24) [141]

As explained in Section 1.1.2, several methods are proposed in the literature to obtain bounds on the end-to-end worst-case delays, given the arrival curve constraints of flows at the sources and service curves offered by the nodes. Finding the best delay bound is an NP-hard problem and is generally not feasible, therefore, several methods were developed to find good delay bounds. Frequently used methods are Total Flow Analysis (TFA) [61, 62, 63, 64], Single Flow Analysis (SFA) [62, Section

3.3], Pay Multiplexing Only Once (PMOO) [62, Section 3.4][65], Least Upper Delay Bound (LUDB) [22, 66, 120, 121], Tandem Matching Analysis (TMA) [176], Polynomial-size Linear Programming (PLP) [68], and Exponential-size Linear Programming (ELP) [20]. All methods provide valid delay bounds but differ in their design and implementation, and it is not trivial to identify the best, smallest bound among them. Therefore, it is interesting to compare different methods and find the smallest delay bound.

The existing worst-case delay analysis tools, such as xTFA [61], DiscoDNC [66, 124], and Panco [136] (see Section 2.1.4 for more tools), support some of the frequently used methods. These tools altogether cover most of the widely recognizable methods within the community. As of today, despite the great potential of utilizing multiple tools, users must implement multiple pieces of code with different syntaxes for each of them, which is impractical and error-prone.

We present Saihu, **S**uperimposed worst-case delay **A**nalysis **I**nterface for **H**uman-friendly **U**sage, to simplify the whole process. Users can execute analyses and compare the results from each tool easily with a single interface and simple commands. Saihu provides a general interface that enables defining the networks in one XML or JSON file and executing all tools simultaneously without any modification; it automatically generates input for each tool respectively and executes the analyses on them. Saihu can produce analysis results in formatted reports and offer automatic network generation for certain types of networks. Therefore, with its straightforward syntax and ease of execution, Saihu simplifies the worst-case bounds comparisons in time-sensitive networks. Its design is modular and supports the addition of new tools. Fig. A.0.1 illustrates the design of Saihu with its data flow. An introductory video is available on https://youtu.be/MiOhLay8Kr4.

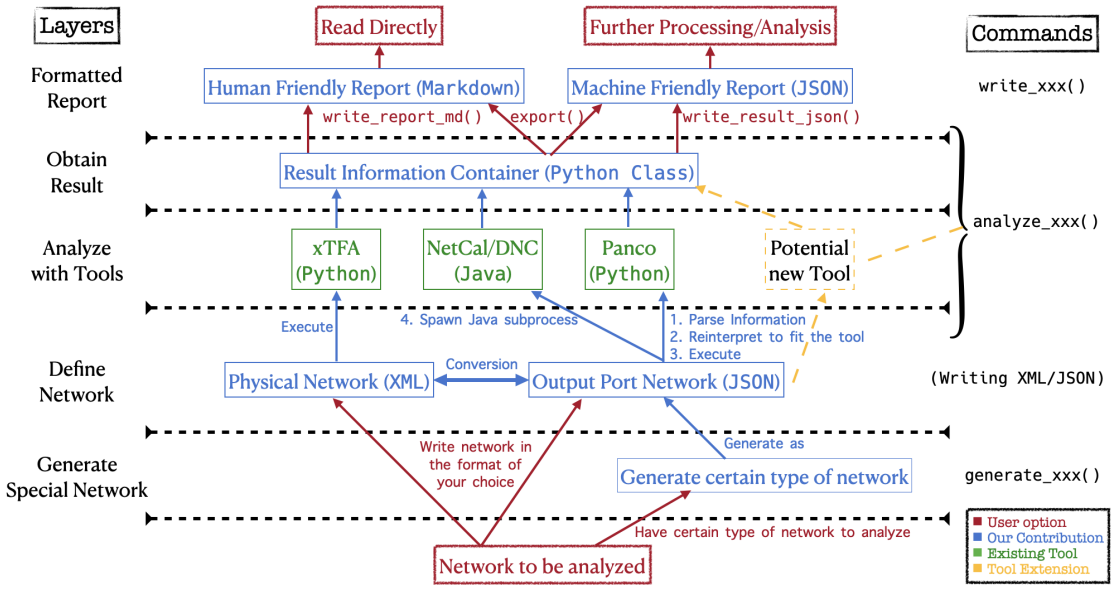


Figure A.0.1: Data flow of Saihu: Red represents the user options; blue is for our contribution; green is for the existing tools; and yellow is for the potential extension of Saihu for more tools. It automates all the programming details in the middle and requires only a few commands listed on the right.

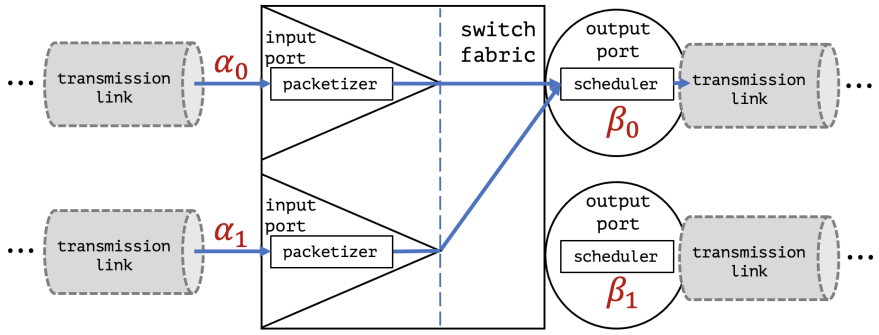


Figure A.1.1: Device model.

A.1 System Model

Devices represent switches or routers that compose the network of interest; they consist of input ports, output ports, and a switching fabric. Fig. A.1.1 shows one such device. Each packet enters a device via an input port and is stored in a packetizer. A packetizer releases a packet only when the entire packet is received. Then, the packet goes through a switching fabric, which transmits the packet to a specific output port based on the static route of the packet; the packet is either buffered in a First-In-First-Out (FIFO) queue and then is serialized on the output line at the transmission rate of the line or exits the network via a terminal port (i.e., a sink).

A flow is a stream of packets generated from the same source, following the same path, and destined for the same sink. We assume that flows are statically assigned to a path. A path consists of a source, a sequence of devices (with the corresponding input port and output port), and a sink (see Fig. A.3.1). For each flow i, we let $l_i^{\max}$ and $l_i^{\min}$ denote the maximum and minimum packet size, respectively. Every flow is constrained at the source by an arrival curve which we assume to be piece-wise linear and concave. Such an arrival curve, say α, can be described by a collection of m rates $r_1, r_2, ..., r_m$ and bursts $b_1, b_2, ..., b_m$ such that $\alpha(t) = \min_{k=1:m}(r_k t + b_k)$; each function $t \mapsto r_k t + b_k$ is called a token-bucket function with rate r_k and burst b_k. The long-term arrival rate of a flow is $\min_k r_k$. Note that the parameters $(m, r_{1:m}, b_{1:m})$ may differ for every flow.

A flow can be either *unicast* (one source, one destination) or *multicast* (one source, multiple destinations). In the latter case, traffic can be split at one or several intermediate devices. For the tools that do not support multicast flows, we replace every multicast flow with p sub-paths by p unicast flows with the same arrival curve constraint at the source; this increases the traffic inside the network, and thus delay bounds that we obtain are valid but might be less good than those obtained by tools that natively support multicast flows.

The service offered to the aggregation of all flows of interest at an output port is represented by a service curve, which we assume to be piece-wise linear and convex. Such a service curve, say β, can be described by a collection of n rates $R_1, R_2, ..., R_n$ and latencies $T_1, T_2, ..., T_n$ such that $\beta(t) = \max_{k=1:n}(R_k[t - T_k]^+)$, with the notation $[x]^+ = \max(x, 0)$; each function $t \to R_k[t - T_k]^+$ is called a rate-latency function with rate R_k and latency T_k. The long-term service rate of the output port is defined as $\max_k R_k$. Note that the parameters $(n, R_{1:n}, T_{1:n})$ may differ at every output port.

We assume that the network is locally stable, namely, at every output port, the aggregate long-term arrival rate (equal to the sum of the long-term arrival rates of all flows using the output port) is less

than the long-term service rate. This is a necessary condition for the existence of finite delay bounds; it is also sufficient in feed-forward networks, but not in networks that have cyclic dependencies [40, Chapter 12].

A.2 Included Tools

Saihu currently includes three tools: xTFA, DNC, and Panco. Fig. A.2.1 summarizes supported methods for each tool.

Method\Tool	DNC	xTFA	Panco
TFA	V	V	V
SFA	V		V
PLP			V
ELP			V
PMOO	V		
LUDB	V		
TMA	V		

Table A.2.1: Supported methods are marked with a “V”.

- **xTFA** [61] is developed in Python and supports a more advanced TFA. For its input, an XML file describes the network (cf. Sec. A.3.1.1). xTFA supports analyzing networks with cyclic dependency and multicast flows.
- **DiscoDNC** [124] is developed in Java and partially uses linear programming with *CPLEX* [138] for LUDB. It supports TFA, SFA, PMOO, LUDB, and TMA. It supports more methods, for example, Unique Linear Program (ULP) [177], but there are not supported yet in Saihu. A network is defined through its own Java classes. Saihu uses the information from an output port network to create a network in DNC syntax internally. Moreover, with DNC, one cannot manually set shaping with FIFO multiplexing but only with arbitrary multiplexing. Also, DNC does not support networks with cyclic dependency and does not support multicast flows (see Section A.1).
- **Panco** [136] is developed in Python and uses linear programming. So, it requires *lpsolve* [139] to execute TFA, SFA, PLP, and ELP. A network is described with its own Python classes. Saihu internally creates the network in Panco syntax from the information of an output port network. All methods of Panco except for ELP support networks with cyclic dependencies. Panco does not support multicast flows (see Section A.1).

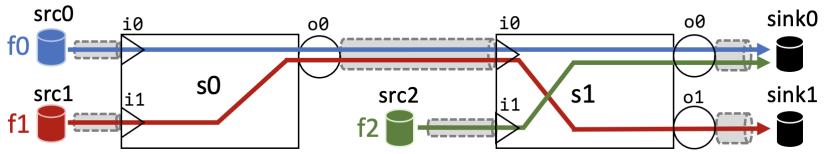
A.3 Software Description

Saihu’s analysis are done in three steps: describe a network to be analyzed (Sec. A.3.1); execute analyses with selected tools (Sec. A.3.2); and export analysis reports back to the user (Sec. A.3.3).

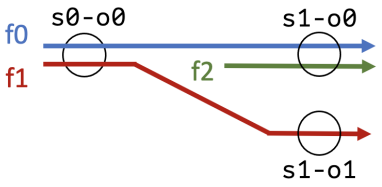
A.3.1 Network Description File

Saihu allows the user to write a network in either a *physical network* or an *output port network* format. Examples are shown in Fig. A.3.1. Briefly speaking, a physical network represents the physical connec-

tions between multiple switches and stations and flows that travel through different input and output ports of switches; it represents the view of a real-world network. On the other hand, as a physical network includes more than enough information, we provide the output port network format as a simplified form to define a network. As we assume the output ports to be the main points of resource competition, even if we provide full network information, we only describe output ports as service units instead of the entire device.



(a) Physical Network.



(b) Output Port Network.

Figure A.3.1: Physical and output port network examples

Although the output port network contains all the necessary information for analysis, we still provide both physical and output port networks as available input file forms. People may prefer to write directly in the physical network format to avoid the translation to an output port format, and some people may prefer the output port network to write a network concisely.

While xTFA takes a physical network as an XML file and the others parse from an output port network as a JSON file, one can choose the format they prefer to define a network as Saihu automatically converts a file when needed.

A.3.1.1 Option 1: Physical Network in XML

A physical network is written as an XML file in the same format as in xTFA, and at least contains **General network information**, **Servers**, **Links**, and **Flows**. A minimal example of defining a physical network is shown in Listing. A.1.

```
<network name="demo" technology="FIFO+IS" minimum-packet-size="50B"/>
<station name="src0"/>
<switch name="s0" service-latency="10us" service-rate="4Mbps"/>
<station name="sink0"/>
<link name="src0-s0" from="src0" to="s0" fromPort="o0" toPort="i0"/>
<link name="s0-sink0" from="s0" to="sink0" fromPort="o0" toPort="i0" transmission-
  capacity="10Mbps"/>
<flow name="f0" arrival-curve="leaky-bucket" lb-burst="10B" lb-rate="10kbps" maximum-
  packet-size="50B" source="src0">
<target>
<path node="s0"/>
<path node="sink0"/>
</target>
</flow>
```

Listing A.1: Example of a physical network representation

First, one `network` element defines the general network information as its attributes: the network's name (`name`), several analysis parameters concatenated by the plus sign (`technology`, `IS` stands for *Input Shaping*), and optionally some default value (e.g. `minimum-packet-size`) across the network.

Second, the servers of the network are either a `station` or a `switch` and represent a physical node. Although they are physically different, they both serve as service-providing devices in our tools, as mentioned in Section A.1, or sources/sinks of a data flow. The service parameters `service-latency` and `service-rate` define a default service curve for all the output ports on this device.

Third, a `link` connects two devices. Saihu tools consider output ports as processing units, so the physical link must be defined `from` a physical node `to` another node with the input and output ports used by the link. Namely, it goes from an output port of one server to an input port of another server. Since a link directly attaches to an output port, users can define service via a link. The `transmission-capacity` of the link can also be specified to consider line shaping. Without defined values, the system will apply the default values defined at the upper levels (`switch`/`station` or `network`).

Finally, a `flow` element defines a flow. Flow paths are surrounded by `target` elements, where each node it traverses is listed as a `path` element with its `node` attribute indicating the name of the physical node. In this format, multicast flow is possible by defining multiple `target` elements within the same flow. A token-bucket curve at the `source` is defined by `arrival-curve`, `lb-burst`, and `lb-rate` keywords. Packetization is considered with maximum and minimum packet sizes. Saihu analyzes all the output ports in the order of the flow path.

A.3.1.2 Option 2: Output Port Network in JSON

Output port format is designed by the authors to write the network concisely. The file contains at least **General network information**, **Servers**, and **Flows**. An example is shown as Listing A.2.

First, a `network` object defines the general network information, the default values, and units throughout the network.

Second, `servers` defines all servers as an array. The parameters can be either a *string* as a number

```
{
    "network": {
        "name": "demo",
        "multiplexing": "FIFO",
        "rate_unit": "Mbps"
    },
    "servers": [
    {
        "name": "s0-o0",
        "service_curve": {
            "latencies": ["10us", "1ms"],
            "rates": [4, "50Mbps"]
        },
        "capacity": "200Mbps"
    }
    ],
    "flows": [
    {
        "name": "f0",
        "path": ["s0-o0"],
        "arrival_curve": {
            "bursts": ["10B", "2kB"],
            "rates": ["10kbps", 0.5]
        },
        "max_packet_length": "50B",
    }
    ]
}
```

Listing A.2: Network information with default values.

followed by a unit, e.g., `"10us"` for 10 microseconds; or a *number* that uses the predefined unit. The service curve is taken as the maximum of all rate-latency curves defined in `service_curve` (see Sec. A.1). Each rate-latency curve is described by a pair of rate and latency values with the same index. For example, the service curve of server `s0-o0` is derived from 2 rate-latency curves: the first has a 10 microseconds latency and 4 megabits per second rate, and the second has a 1000 microseconds latency and 50 megabits per second rate.

Notice that in an output port network format, we don't manually define links. We use *graph-induced-by-flows* as the network topology. A link between two servers exists only when at least one flow crosses these two servers consecutively. Therefore, the link's transmission capacity attached to an output port is directly defined on a server with the keyword `capacity`.

Finally, `flows` represents the flows as an array. Each flow is defined by a `path` as an array of servers, and an `arrival_curve` at its source. The arrival curve is defined as the minimum of the multiple token-bucket curves, each pair of burst and rate values represents a token-bucket curve (see Section A.1). For example, `f0`'s arrival curve is composed of a token-bucket curve of burst 10 bytes and rate 10 kbps, and a curve of burst 2 kilobytes and rate 0.5 megabits per second.

All flows written in the output port network format are assumed to be unicast flows. When being converted from a physical network with multicast flows, it separates the paths into multiple unicast flows with the same source and arrival curve (see Section A.1).

```
python main.py demo.json -a
```

Listing A.3: Use Saihu as command line tool.

```
from saihu.interface import TSN_Analyzer
analyzer = TSN_Analyzer("demo.json")
analyzer.analyze_all()
analyzer.export("demo")
```

Listing A.4: Use Saihu as package.

A.3.2 Tool Usage

Saihu analysis execution can be done via the command line tool `main.py` or by importing the file `interface.py` if one wishes to integrate Saihu into their project. We demonstrate the simplest way to analyze a network file, say *demo.json*, with both possibilities. Listing A.3 and Listing A.4 show two ways to analyze *demo.json* with all the tools and methods available inside Saihu.

To switch between different tools in the package, one uses different functions with names like `analyze_xxx`, where `analyze_all` executes all available tools. Methods are specified as function arguments (cf. Listing A.5). One can execute multiple analyses and all the results will be stored in the internal buffer of the analyzer until they are exported into Saihu reports. As for the command line interface, selecting tools or methods is simply specifying different flags.

```
analyzer.analyze_dnc("LUDB")
analyzer.analyze_panco(methods=["SFA", "PLP"])
```

Listing A.5: Execute different tools and methods.

A.3.3 Analysis Reports

Saihu can generate 2 kinds of reports:

1. A *human-friendly report* is generated as a Markdown file that gives the per-flow end-to-end delay, per-server delay, and execution time for each tool. The delay bounds are presented in tables where each row is a flow or a server, and each column is a method executed by a tool. The last column contains the minimum result obtained in the current round of analysis. Fig. A.3.2 demonstrates an example of per-flow end-to-end delay and execution time as a reference.

   The report also contains some reminders about the user inputs: network topology using the graph-induced-by-flows (Sec. A.3.1.2), flow paths, and link utilization by nodes. Link utilization is defined as the ratio between the aggregated arrival rate at a node and its service rate.

2. A *machine-friendly report* is written in JSON format for easy parsing from other programs. It stores the execution outputs, i.e. the per-flow end-to-end delay, per-server delay, and execution time. An example is shown in Listing A.6. Note that the numbers in a human-friendly report are always rounded to 3 decimal digits while there's no such rounding for a machine-friendly report. As a result, one should read the machine-friendly report if they require a very precise result.

Unit in microsecond

Flow name	Panco-ELP	DNC-LUDB	Panco-PLP	DNC-PMOO	DNC-SFA	xTFA-TFA	Minimum (best)
f0	80.050	80.050	80.050	80.201	80.050	99.324	80.050
f1	80.080	60.050	79.270	60.125	60.050	79.322	60.050
f2	49.280	50.004	49.280	50.201	50.125	49.324	49.280

(a) Flow end-to-end delay.

Unit in millisecond

tool-method	Execution Time
Panco-ELP	123.159
DNC-LUDB	202.000
Panco-PLP	139.048
DNC-PMOO	16.000
DNC-SFA	13.000
xTFA-TFA	5.334

(b) Execution time.

Figure A.3.2: Human-friendly Markdown report.

```
{
    "name": "demo",
    "flow_e2e_delay": {
        "f0": {
            "xTFA_TFA": 99.32394489448944,
            "Panco_PLP": 80.05,
            "Panco_ELP": 80.05,
            "DNC_SFA": 80.0501253132832,
            "DNC_PMOO": 80.20050125313283,
            "DNC_LUDB": 80.0501253132832
        },
        ...
        "server_delay": {
            "s0-o0": {
                "xTFA_TFA": 50.0
            },
            ...
            "execution_time": {
                "xTFA_TFA": 5.716085433959961,
                "Panco_PLP": 147.26519584655762,
                "Panco_ELP": 129.76408004760742,
                "DNC_SFA": 12.0,
                "DNC_PMOO": 9.0,
                "DNC_LUDB": 172.0
            },
            "units": {
                "flow_delay": "us",
                "server_delay": "us",
                "execution_time": "ms"
            }
        }
```

Listing A.6: JSON report.

A.4 Conclusion and Extension

We presented Saihu, a Python interface capable of executing multiple network analysis tools easily. It allows users to define a network and retrieve the analysis results for multiple tools. Its unified input and

output layers enable the automation of all the tedious re-interpretation of a network for each tool. Such design not only simplifies many mechanical procedures previously hindering network researchers' works but also helps publicize these useful tools to more potential users due to its simplicity.

A.5 Current code version

Nr.	Code metadata description	Information
C1	Current code version	v1
C2	Permanent link to code/repository used for this code version	https://github.com/adfeel220/Saihu-TSN-Analysis-Tool-Integration
C3	Permanent link to Reproducible Capsule	
C4	Legal Code License	MIT License
C5	Code versioning system used	Git
C6	Software code languages, tools, and services used	Python, Java (by DiscoDNC), lpsolve (by Panco), and CPLEX (by *LUDB* option of DiscoDNC)
C7	Compilation requirements, operating environments & dependencies	Python packages: `xTFA`, `Panco`, Python packages `numpy`, `networkx`, `matplotlib`, `mdutils`, and `pulp`. Java package: `DiscoDNC`
C8	If available Link to developer documentation/manual	https://github.com/adfeel220/Saihu-TSN-Analysis-Tool-Integration/blob/main/README.md
C9	Support email for questions	chun-tso.tsai@epfl.ch

Table A.5.1: Code metadata

Bibliography

[1] David D. Walden, Garry J. Roedler, Kevin Forsberg, R. Douglas Hamelin, and Thomas M. Shortell. *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. Wiley, Hoboken, NJ, fourth edition, 2015. ISBN 978-1118999400.

[2] IEEE 802.1 Working Group. Use Cases - IEEE P802.1DG, V0.4, 2019. URL https://www.ieee802.org/1/files/public/docs2019/dg-pannell-automotive-use-cases-0919-v04.pdf. [Online].

[3] IEEE 802.1 Working Group. Aerospace Traffic Characterization, 2021. URL https://www.ieee802.org/1/files/public/docs2021/dp-Jabbar-et-all-Aerospace-Traffic-Characterization-0421-v02.pdf. [Online].

[4] IEEE 802.1 Working Group. Industrial Automation Traffic Types and their Mapping to QoS/TSN Mechanisms, 2018. URL https://www.ieee802.org/1/files/public/docs2018/60802-ademaj-traffic-type-characterization-1118-v01.pdf. [Online].

[5] Ian F. Akyildiz and Josep M. Jornet. The tactile internet: Vision, recent progress, and open challenges. *IEEE Internet of Things Journal*, 3(5):596–616, 2016. doi: 10.1109/JIOT.2016.2576398.

[6] ITU-T. ITU-T Y.3000-series - Representative use cases and key network requirements for Network 2030. Technical report, 2020. URL https://www.itu.int/rec/T-REC-Y.Sup67-202007-I.

[7] T. Wong, N. Finn, and X. Wang. TSN Profile for Service Provider Networks, No Date. URL https://www.ieee802.org/1/files/public/docs2018/new-tsn-wangtt-TSN-profile-for-service-provider-network-0718.pdf.

[8] C. Mannweiler, B. Gajic, P. Rost, R. S. Ganesan, C. Markwart, R. Halfmann, J. Gebert, and A. Wich. Reliable and deterministic mobile communications for industry 4.0: Key challenges and solutions for the integration of the 3gpp 5g system with ieee. In *Mobile Communication - Technologies and Applications; 24. ITG-Symposium*, pages 1–6, 2019.

[9] ISO/IEC/IEEE. ISO/IEC/IEEE International Standard - Information technology – Telecommunications and information exchange between systems – Local and Metropolitan Area Networks – Specific requirements – Part 1BA: Audio video bridging (AVB) systems, 2016.

[10] Time-sensitive networking (TSN) task group. Time-sensitive networking (TSN) task group. URL https://1.ieee802.org/tsn/.

[11] Sergiu S. Craciunas, Regina S. Oliver, and Tarun Ag. An overview of scheduling mechanisms for time-sensitive networks. In *Proceedings of the French Summer School on Real-Time Systems l'École d'Été Temps Réel (ETR)*, pages 1551–3203, Gif-sur-Yvette, France, 2017.

[12] Deterministic Networking (DetNet). URL https://datatracker.ietf.org/wg/detnet/about/. Accessed: 2019-03-26.

[13] E. Grossman. RFC8578: Deterministic Networking Use Cases, May 2019. URL https://www.rfc-editor.org/info/rfc8578.

[14] P802.1DG – TSN Profile for Automotive In-Vehicle Ethernet Communications. https://1.ieee802.org/tsn/802-1dg/, N/A. Accessed: 13/07/2022.

[15] IEC and IEEE. IEC/IEEE 60802 - Time-Sensitive Networking Profile for Industrial Automation. Iec/ieee 60802 (d1.3), Sep. 2021.

[16] IEEE. Draft Standard for Local and metropolitan area networks — Time-Sensitive Networking Profile for Service Provider Networks. Ieee p802.1df™/d0.1, Dec. 2020.

[17] IEEE. IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks. IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014), Jul. 2018. Conference Name: IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014).

[18] R. Salazar, T. Godfrey, N. Finn, C. Powell, B. Rolfe, and M. Seewald. Utility Applications of Time Sensitive Networking White Paper. Technical report, Utility Applications of Time Sensitive Networking White Paper, 2019.

[19] N. Finn, P. Thubert, B. Varga, and J. Farkas. RFC8655: Deterministic Networking Architecture. Technical report, 2019. URL https://www.rfc-editor.org/info/rfc8655.

[20] Anne Bouillard and Giovanni Stea. Exact worst-case delay in fifo-multiplexing feed-forward networks. *IEEE/ACM Transactions on Networking*, 23(5):1387–1400, 2015. doi: 10.1109/TNET.2014.2332071.

[21] Anne Bouillard and Aurore Junier. Worst-case delay bounds with fixed priorities using network calculus. In *Proceedings of the 5th International ICST Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS '11, page 381–390, Brussels, BEL, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 9781936968091.

[22] Alexander Scheffler and Steffen Bondorf. Network calculus for bounding delays in feedforward networks of fifo queueing systems. In Alessandro Abate and Andrea Marin, editors, *Quantitative Evaluation of Systems*, pages 149–167, Cham, 2021. Springer International Publishing. ISBN 978-3-030-85172-9.

[23] H. Charara, J.-L. Scharbarg, J. Ermont, and C. Fraboul. Methods for bounding end-to-end delays on an afdx network. In *18th Euromicro Conference on Real-Time Systems (ECRTS'06)*, pages 10 pp.–202, 2006. doi: 10.1109/ECRTS.2006.15.

[24] Jonathan Falk, David Hellmanns, Ben Carabelli, Naresh Nayak, Frank Dürr, Stephan Kehrer, and Kurt Rothermel. Nesting: Simulating ieee time-sensitive networking (tsn) in omnet++. In *2019 International Conference on Networked Systems (NetSys)*, pages 1–8, 2019. doi: 10.1109/NetSys.2019.8854500.

[25] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, apr 1986. ISSN 0164-0925. doi: 10.1145/5397.5399. URL https://doi.org/10.1145/5397.5399.

[26] J. Krakora, L. Waszniowski, P. Pisa, and Z. Hanzalek. Timed automata approach to real time distributed system verification. In *IEEE International Workshop on Factory Communication Systems, 2004. Proceedings.*, pages 407–410, 2004. doi: 10.1109/WFCS.2004.1377759.

[27] Daniel Witsch, Birgit Vogel-Heuser, Jean-Marc Faure, and Gaëlle Marsal. Performance analysis of industrial ethernet networks by means of timed model-checking. *IFAC Proceedings Volumes*, 39 (3):101–106, 2006. ISSN 1474-6670. doi: https://doi.org/10.3182/20060517-3-FR-2903.00063. URL https://www.sciencedirect.com/science/article/pii/S1474667015357839. 12th IFAC Symposium on Information Control Problems in Manufacturing.

[28] Steven Martin. *Maîtrise de la dimension temporelle de la qualité de service dans les réseaux*. Theses, Université Paris XII Val de Marne, July 2004. URL https://theses.hal.science/tel-00007638. Francis COTTET (ENSMA)
Françoise SIMONOT-LION (ENSMN)
Yacine AMIRAT(Université Paris 12)
Laurent GEORGE (Université Paris 12)
Pascal LORENZ (Université de Haute Alsace)
Pascale MINET (INRIA Rocquencourt
Samir TOHME (Université de Versailles St-Quentin).

[29] Xiaoting Li, Olivier Cros, and Laurent George. The Trajectory approach for AFDX FIFO networks revisited and corrected. In *The 20th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Chongqing, China, August 2014. IEEE. URL https://hal.science/hal-00975730.

[30] Henri Bauer, Jean-Luc Scharbarg, and Christian Fraboul. Applying Trajectory approach with static priority queuing for improving the use of available AFDX resources. In *18th International Conference on Real-Time and Network Systems*, pages 69–78, Toulouse, France, November 2010. URL https://hal.science/hal-00544508.

[31] H. Charara, J. . Scharbarg, J. Ermont, and C. Fraboul. Methods for bounding end-to-end delays on an afdx network. In *18th Euromicro Conference on Real-Time Systems (ECRTS'06)*, pages 10 pp.–202, 2006. doi: 10.1109/ECRTS.2006.15.

[32] Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40(2):117–134, 1994. ISSN 0165-6074. doi: https://doi.org/10.1016/0165-6074(94)90080-9. URL https://www.sciencedirect.com/science/article/pii/0165607494900809. Parallel Processing in Embedded Real-time Systems.

[33] Paul Pop, Petru Eles, and Zebo Peng. Scheduling with optimized communication for time-triggered embedded systems. In *Proceedings of the seventh international workshop on hardware/-software codesign*, pages 178–182, 1999.

[34] P. Pop, P. Eles, and Zebo Peng. Schedulability analysis and optimization for the synthesis of multi-cluster distributed embedded systems. In *2003 Design, Automation and Test in Europe Conference and Exhibition*, pages 184–189, 2003. doi: 10.1109/DATE.2003.1253606.

[35] T. Pop, P. Pop, P. Eles, and Zebo Peng. Optimization of hierarchically scheduled heterogeneous embedded systems. In *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*, pages 67–71, 2005. doi: 10.1109/RTCSA.2005.67.

[36] Domitian Tamas-Selicean, Paul Pop, and Wilfried Steiner. Synthesis of communication schedules for ttethernet-based mixed-criticality systems. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS

'12, page 473–482, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450314268. doi: 10.1145/2380445.2380518. URL https://doi.org/10.1145/2380445.2380518.

[37] Domiţian Tămaş-Selicean, Paul Pop, and Wilfried Steiner. Design optimization of ttethernet-based distributed real-time systems. *Real-Time Systems*, 51(1):1–35, 2015. doi: 10.1007/s11241-014-9214-8. URL https://doi.org/10.1007/s11241-014-9214-8.

[38] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, volume 2050. Springer Science & Business Media, 2001. ISBN 978-3-540-42184-9.

[39] C. S. Chang. *Performance Guarantees in Communication Networks*. Springer-Verlag, New York, 2000.

[40] Anne Bouillard, Marc Boyer, and Euriell Le Corronc. *Deterministic Network Calculus: From Theory to Practical Implementation*. Wiley-ISTE, 2018. ISBN 978-1-119-56341-9.

[41] Michel Boyer and Christian Fraboul. Tightening end to end delay upper bound for AFDX network calculus with rate latency FIFO servers using network calculus. In *Proceedings of the 2008 IEEE International Workshop on Factory Communication Systems*, pages 11–20, Dresden, Germany, 2008. IEEE.

[42] Jérôme Grieu. Analyse et évaluation de techniques de commutation ethernet pour l'interconnexion des systèmes avioniques. September 2004. URL https://oatao.univ-toulouse.fr/7385/.

[43] Fabrice Frances, Christian Fraboul, and Jérôme Grieu. Using network calculus to optimize the AFDX network. 2006.

[44] Luyue Ji, Wenjie Wu, Chaojie Gu, Jichao Bi, Shibo He, and Zhiguo Shi. Network calculus-based routing and scheduling in software-defined industrial internet of things. In *2022 IEEE 20th International Conference on Industrial Informatics (INDIN)*, pages 463–468, 2022. doi: 10.1109/INDIN51773.2022.9976177.

[45] Yinzhi Lu, Liu Yang, Simon X. Yang, Qiaozhi Hua, Arun Kumar Sangaiah, Tan Guo, and Keping Yu. An intelligent deterministic scheduling method for ultralow latency communication in edge enabled industrial internet of things. *IEEE Transactions on Industrial Informatics*, 19(2):1756–1767, 2023. doi: 10.1109/TII.2022.3186891.

[46] Rahul Nandkumar Gore, Elena Lisova, Johan Åkerberg, and Mats Björkman. Network calculus approach for packet delay variation analysis of multi-hop wired networks. *Applied Sciences*, 12(21), 2022. ISSN 2076-3417. doi: 10.3390/app122111207. URL https://www.mdpi.com/2076-3417/12/21/11207.

[47] Qian Ren, Kui Liu, and Lianming Zhang. Multi-objective optimization for task offloading based on network calculus in fog environments. *Digital Communications and Networks*, 8(5):825–833, 2022. ISSN 2352-8648. doi: https://doi.org/10.1016/j.dcan.2021.09.012. URL https://www.sciencedirect.com/science/article/pii/S2352864821000729.

[48] G. Kesidis, Y. Shan, B. Urgaonkar, and J. Liebeherr. Network calculus for parallel processing. *SIGMETRICS Perform. Eval. Rev.*, 43(2):48–50, sep 2015. ISSN 0163-5999. doi: 10.1145/2825236.2825256. URL https://doi.org/10.1145/2825236.2825256.

[49] Nicholas Jacobs, Shamina Hossain-McKenzie, and Adam Summers. Modeling data flows with network calculus in cyber-physical systems: Enabling feature analysis for anomaly detection applications. *Information*, 12(6), 2021. ISSN 2078-2489. doi: 10.3390/info12060255. URL https://www.mdpi.com/2078-2489/12/6/255.

[50] Huan Yang, Liang Cheng, and Xiaoguang Ma. Combining measurements and network calculus in worst-case delay analyses for networked cyber-physical systems. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1065–1066, 2019. doi: 10.1109/INFCOMW.2019.8845285.

[51] Huan Yang, Liang Cheng, and Xiaoguang Ma. Combining measurements and network calculus in worst-case delay analyses for networked cyber-physical systems. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1065–1066, 2019. doi: 10.1109/INFCOMW.2019.8845285.

[52] Sheng Zhu, Zhen Sun, Yong Lu, Lianming Zhang, Yehua Wei, and Geyong Min. Centralized qos routing using network calculus for sdn-based streaming media networks. *IEEE Access*, 7: 146566–146576, 2019. doi: 10.1109/ACCESS.2019.2943518.

[53] Joan Adrià Ruiz De Azua and Marc Boyer. Complete modelling of avb in network calculus framework. In *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, RTNS '14, page 55–64, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450327275. doi: 10.1145/2659787.2659810. URL https://doi.org/10.1145/2659787.2659810.

[54] Anne Bouillard, Marc Boyer, and Euriell Le Corronc. *Deterministic Network Calculus: From Theory to Practical Implementation*. John Wiley & Sons, 2018.

[55] Hugo Daigmorte, Marc Boyer, and Luxi Zhao. Modelling in network calculus a TSN architecture mixing Time-Triggered, Credit Based Shaper and Best-Effort queues. working paper or preprint, June 2018. URL https://hal.science/hal-01814211.

[56] Kohei Hirano and Yoshihiro Ito. Study on appropriate idleslope value of credit based shaper for qos control on in-vehicle ethernet. In *2020 IEEE 9th Global Conference on Consumer Electronics (GCCE)*, pages 686–687, 2020. doi: 10.1109/GCCE50665.2020.9291833.

[57] Ehsan Mohammadpour, Eleni Stai, Maaz Mohiuddin, and Jean-Yves 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)*, volume 02, pages 1–6, 2018. doi: 10.1109/ITC30.2018.10053.

[58] Luxi Zhao, Paul Pop, Zhong Zheng, and Qiao Li. Timing analysis of avb traffic in tsn networks using network calculus. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 25–36, 2018. doi: 10.1109/RTAS.2018.00009.

[59] Dinh-Khanh Dang and Ahlem Mifdaoui. Timing analysis of tdma-based networks using network calculus and integer linear programming. In *2014 IEEE 22nd International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 21–30, 2014. doi: 10.1109/MASCOTS.2014.12.

[60] Ehsan Mohammadpour, Eleni Stai, and Jean-Yves Le Boudec. Improved network calculus delay bounds in time-sensitive networks, 2022.

[61] Ludovic Thomas. *Analysis of the side-effects on latency bounds of combinations of scheduling, redundancy and synchronization mechanisms in time-sensitive networks*. PhD thesis, l'Institut Supérieur de l'Aéronautique et de l'Espace (ISAE-SUPAERO), 2022. URL http://www.theses.fr/2022ESAE0041. Thèse de doctorat dirigée par Mifdaoui, Ahlem et Le Boudec, Jean-Yves Informatique et Télécommunications Toulouse, ISAE 2022.

[62] Jens B. Schmitt and Frank 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*, valuetools '06, page 8–es, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595935045. doi: 10.1145/1190095.1190105. URL https://doi.org/10.1145/1190095.1190105.

[63] Ahlem Mifadoui and Thierry Leydier. Beyond the Accuracy-Complexity Tradeoffs of CompositionalAnalyses using Network Calculus for Complex Networks. In *10th International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (co-located with RTSS 2017)*, pages pp. 1–8, Paris, France, December 2017. URL https://hal.archives-ouvertes.fr/hal-01690096.

[64] Stephan Plassart and Jean-Yves Le Boudec. Equivalent versions of total flow analysis. *CoRR*, abs/2111.01827, 2021. URL https://arxiv.org/abs/2111.01827.

[65] Jens B. Schmitt, Frank A. Zdarsky, and Ivan Martinovic. Improving performance bounds in feedforward networks by paying multiplexing only once. In *Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen*, 2011.

[66] Alexander Scheffler and Steffen Bondorf. Network calculus for bounding delays in feedforward networks of FIFO queueing systems. In *Proc. of the 18th International Conference on Quantitative Evaluation of Systems*, QEST '21, pages 149–167, August 2021. URL https://link.springer.com/chapter/10.1007/978-3-030-85172-9_8.

[67] Fabien Geyer, Alexander Scheffler, and Steffen Bondorf. Tightening network calculus delay bounds by predicting flow prolongations in the fifo analysis. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 157–170, 2021. doi: 10.1109/RTAS52030.2021.00021.

[68] Anne Bouillard. Trade-off between accuracy and tractability of network calculus in FIFO networks. *Perform. Eval.*, 153(C), feb 2022. ISSN 0166-5316. doi: 10.1016/j.peva.2021.102250. URL https://doi.org/10.1016/j.peva.2021.102250.

[69] Iec 62439: High availability automation networks: High availability automation networks. Technical report, 2012.

[70] Iec 62439-3, industrial communication networks - high availability automation networks - part 3: Parallel redundancy protocol (prp) and high-availability seamless redundancy (hsr). Technical report, 2016.

[71] Ellen L. Hahne and Robert G. Gallager. Round robin scheduling for fair flow control in data communication networks. In *Proc. of the IEEE Int. Conf. on Communications (ICC 86)*, June 1986.

[72] J. Nagle. On packet switches with infinite storage. *Communications, IEEE Transactions on*, 35(4): 435–438, Apr 1987. ISSN 0090-6778. doi: 10.1109/TCOM.1987.1096782.

[73] IEEE standard for local and metropolitan area networks – bridges and bridged networks. IEEE Standard 802.1Q, IEEE, 2018.

[74] Dhinesh Babu L.D. and P. Venkata Krishna. Honey bee behavior inspired load balancing of tasks in cloud computing environments. *Applied Soft Computing*, 13(5):2292–2303, 2013. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2013.01.025. URL http://www.sciencedirect.com/science/article/pii/S1568494613000446.

[75] Wensong. Weighted round-robin scheduling, documentation of the linuxvirtualserver knowledge base, 2005.

[76] Yue Qian, Zhonghai Lu, and Wenhua Dou. Analysis of worst-case delay bounds for best-effort communication in wormhole networks on chip. In *Proc. of the 3rd ACM/IEEE International Symposium on Networks-on-Chip (NoCS 2009)*, pages 44–53. IEEE, 2009.

[77] Ibrahim Afolabi, Tarik Taleb, Konstantinos Samdanis, Adlen Ksentini, and Hannu Flinck. Network slicing and softwarization: A survey on principles, enabling technologies, and solutions. *IEEE Communications Surveys & Tutorials*, 20(3):2429–2453, 2018. doi: 10.1109/COMST.2018.2815638.

[78] Xin Li, Mohammed Samaka, H. Anthony Chan, Deval Bhamare, Lav Gupta, Chengcheng Guo, and Raj Jain. Network slicing for 5g: Challenges and opportunities. *IEEE Internet Computing*, 21 (5):20–27, 2017. doi: 10.1109/MIC.2017.3481355.

[79] Xenofon Foukas, Georgios Patounas, Ahmed Elmokashfi, and Mahesh K. Marina. Network slicing in 5g: Survey and challenges. *IEEE Communications Magazine*, 55(5):94–100, 2017. doi: 10.1109/MCOM.2017.1600951.

[80] L. Lenzini, E. Mingozzi, and G. Stea. Aliquem: a novel DRR implementation to achieve better latency and fairness at $O(1)$ complexity. In *IEEE 2002 Tenth IEEE International Workshop on Quality of Service (Cat. No.02EX564)*, pages 77–86, 2002.

[81] Xin Yuan and Zhenhai Duan. Fair round-robin: A low complexity packet schduler with proportional and worst-case fairness. *IEEE Transactions on Computers*, 58(3):365–379, 2009. doi: 10.1109/TC.2008.176.

[82] Yi-Mao Hsiao, Ming-Jen Chen, Yier Chen, Yuan-Sun Chu, and Cheng-Shong Wu. Design and implementation of pipelined drr asic. In *2008 14th Asia-Pacific Conference on Communications*, pages 1–4, 2008.

[83] Linhua Zhong, Jin Xu, and Xianlei Wang. Vwqgrr: A novel packet scheduler. In *Sixth International Conference on Networking (ICN'07)*, pages 36–36, 2007. doi: 10.1109/ICN.2007.103.

[84] Manolis Katevenis, Stefanos Sidiropoulos, and Costas Courcoubetis. Weighted round-robin cell multiplexing in a general-purpose ATM switch chip. *IEEE Journal on Selected Areas in Communications*, 9(8):1265–1279, 1991.

[85] Rainer Handel, Manfred N. Huber, Stefan Schroder, and Lars Wolf. *ATM Networks: Concepts, Protocols, Applications*. Addison-Wesley Professional, 2001.

[86] S. S. Kanhere and H. Sethu. On the latency bound of deficit round robin. In *Proceedings. Eleventh International Conference on Computer Communications and Networks*, pages 548–553, 2002.

[87] Dimitrios Stiliadis. *Traffic Scheduling in Packet-Switched Networks: Analysis, Design, and Implementation*. PhD thesis, 1996. AAI9637506.

[88] L. Lenzini, E. Mingozzi, and G. Stea. Full exploitation of the deficit round robin capabilities by efficient implementation and parameter tuning.

[89] M. Boyer, G. Stea, and W. M. Sofack. Deficit round robin with network calculus. In *6th International ICST Conference on Performance Evaluation Methodologies and Tools*, pages 138–147, 2012.

[90] A. Soni, X. Li, J. Scharbarg, and C. Fraboul. Optimizing network calculus for switched ethernet network with deficit round robin. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 300–311, 2018.

[91] Anne Bouillard. Individual service curves for bandwidth-sharing policies using network calculus. *IEEE Networking Letters*, 3(2):80–83, 2021. doi: 10.1109/LNET.2021.3067766.

[92] RealTime-at-Work online Min-Plus interpreter for Network Calculus. https://www.realtimeatwork.com/minplus-playground. Accessed: year-month-day.

[93] Raffaele Zippo and Giovanni Stea. Nancy: an efficient parallel network calculus library, 2022. ISSN 2352-7110. URL https://arxiv.org/abs/2205.11449.

[94] Steffen Bondorf and Jens B. Schmitt. The DiscoDNC v2 – a comprehensive tool for deterministic network calculus. In *Proc. of the International Conference on Performance Evaluation Methodologies and Tools*, ValueTools '14, pages 44–49, December 2014. URL https://dl.acm.org/citation.cfm?id=2747659.

[95] Urban Suppiger, Simon Perathoner, Kai Lampka, and Lothar Thiele. Modular performance analysis of large-scale distributed embedded systems: An industrial case study. Report, Zurich, 2010-11.

[96] Anne Bouillard and Thomas Nowak. Fast symbolic computation of the worst-case delay in tandem networks and applications. *Perform. Eval.*, 91:270–285, 2015. doi: 10.1016/j.peva.2015.06.016.

[97] Markus Fidler and Amr Rizk. A guide to the stochastic network calculus. *IEEE Communications Surveys & Tutorials*, 17(1):92–105, 2015. doi: 10.1109/COMST.2014.2337060.

[98] Felix Poloczek and Florin Ciucu. Scheduling analysis with martingales. *Performance Evaluation*, 79:56–72, 2014. ISSN 0166-5316. doi: https://doi.org/10.1016/j.peva.2014.07.004. URL http://www.sciencedirect.com/science/article/pii/S0166531614000674. Special Issue: Performance 2014.

[99] M. Vojnovic and J.-Y. Le Boudec. Bounds for independent regulated inputs multiplexed in a service curve network element. In *GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No.01CH37270)*, volume 3, pages 1857–1861 vol.3, 2001. doi: 10.1109/GLOCOM.2001.965896.

[100] Cheng-Shang Chang, Yuh-ming Chiu, and Wheyming Tina Song. On the performance of multiplexing independent regulated inputs. In *Proceedings of the 2001 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '01, page

184–193, New York, NY, USA, 2001. Association for Computing Machinery. ISBN 1581133340. doi: 10.1145/378420.378782. URL https://doi.org/10.1145/378420.378782.

[101] Florin Ciucu and Jens Schmitt. Perspectives on network calculus: No free lunch, but still good value. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, page 311–322, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450314190. doi: 10.1145/2342356.2342426. URL https://doi.org/10.1145/2342356.2342426.

[102] Fabrice M. Guillemin, Ravi R. Mazumdar, Catherine P. Rosenberg, and Yu Ying. A stochastic ordering property for leaky bucket regulated flows in packet networks. *Journal of Applied Probability*, 44(2):332–348, 2007. ISSN 00219002. URL http://www.jstor.org/stable/27595845.

[103] G. Kesidis and T. Konstantopoulos. Worst-case performance of a buffer with independent shaped arrival processes. *IEEE Communications Letters*, 4(1):26–28, 2000. doi: 10.1109/4234.823539.

[104] Seyed Mohammadhossein Tabatabaee, Jean-Yves Le Boudec, and Marc Boyer. Interleaved weighted round-robin: A network calculus analysis. In *2020 32nd International Teletraffic Congress (ITC 32)*, pages 64–72, 2020. doi: 10.1109/ITC3249928.2020.00016.

[105] Seyed Mohammadhossein TABATABAEE, Jean-Yves LE BOUDEC, and Marc BOYER. Interleaved weighted round-robin: A network calculus analysis. *IEICE Transactions on Communications*, E104.B(12):1479–1493, 2021. doi: 10.1587/transcom.2021ITI0001.

[106] Vlad-Cristian Constantin, Paul Nikolaus, and Jens Schmitt. Improving performance bounds for weighted round-robin schedulers under constrained cross-traffic. In *2022 IFIP Networking Conference (IFIP Networking)*, pages 1–9, 2022. doi: 10.23919/IFIPNetworking55013.2022.9829772.

[107] Seyed Mohammadhossein Tabatabaee and Jean-Yves Le Boudec. Deficit round-robin: A second network calculus analysis. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 171–183, 2021. doi: 10.1109/RTAS52030.2021.00022.

[108] Seyed Mohammadhossein Tabatabaee and Jean-Yves Le Boudec. Deficit round-robin: A second network calculus analysis. *IEEE/ACM Transactions on Networking*, pages 1–15, 2022. doi: 10.1109/TNET.2022.3164772.

[109] Seyed Mohammadhossein Tabatabaee, Anne Bouillard, and Jean-Yves Le Boudec. Worst-case delay analysis of time-sensitive networks with deficit round-robin, 2022. URL https://arxiv.org/abs/2208.11400.

[110] Seyed Mohammadhossein Tabatabaee, Marc Boyer, Jean-Yves Le Boudec, and Jörn Migge. Efficient and accurate handling of periodic flows in time-sensitive networks. In *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2023. URL http://infoscience.epfl.ch/record/302640.

[111] Seyed Mohammadhossein Tabatabaee, Anne Bouillard, and Jean-Yves Le Boudec. Quasi-deterministic burstiness bound for aggregate of independent, periodic flows, 2023. URL https://arxiv.org/abs/2305.14946.

[112] Github project: Saihu: A common interface of worst-case delay analysis tools for time-sensitive networks. https://github.com/adfeel220/Saihu-TSN-Analysis-Tool-Integration.

[113] Chun-Tso Tsai, Seyed Mohammadhossein Tabatabaee, Stéphan Plassart, and Jean-Yves Le Boudec. Saihu: A common interface of worst-case delay analysis tools for time-sensitive networks, 2023. URL https://arxiv.org/abs/2303.14565.

[114] Jean-Yves Le Boudec. Network calculus made easy. 1996.

[115] Cheng-Shang Chang. A filtering theory for deterministic traffic regulation. In *Proceedings of INFOCOM '97*, volume 2, pages 436–443 vol.2, 1997. doi: 10.1109/INFCOM.1997.644492.

[116] R.L. Cruz. A calculus for network delay. i. network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, 1991. doi: 10.1109/18.61109.

[117] R.L. Cruz. A calculus for network delay. ii. network analysis. *IEEE Transactions on Information Theory*, 37(1):132–141, 1991. doi: 10.1109/18.61110.

[118] Jean-Yves Le Boudec. An introduction to network calculus. YouTube video, Apr. 5 2019. URL https://www.youtube.com/watch?v=ABQ327BTc_o.

[119] Ludovic Thomas, Jean-Yves Le Boudec, and Ahlem Mifdaoui. On cyclic dependencies and regulators in time-sensitive networks. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 299–311, 2019. doi: 10.1109/RTSS46320.2019.00035.

[120] Luciano Lenzini, Linda Martorini, Enzo Mingozzi, and Giovanni Stea. Tight end-to-end per-flow delay bounds in fifo multiplexing sink-tree networks. *Performance Evaluation*, 63(9):956–987, 2006. ISSN 0166-5316. doi: https://doi.org/10.1016/j.peva.2005.10.003. URL https://www.sciencedirect.com/science/article/pii/S0166531605001537.

[121] Luciano Lenzini, Enzo Mingozzi, and Giovanni Stea. End-to-end delay bounds in fifo-multiplexing tandems. In *Proceedings of the 2nd International Conference on Performance Evaluation Methodologies and Tools*, ValueTools '07, Brussels, BEL, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 9789639799004.

[122] Steffen Bondorf. Better bounds by worse assumptions — improving network calculus accuracy by adding pessimism to the network model. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–7, 2017. doi: 10.1109/ICC.2017.7996996.

[123] Github project: Open-source implementation of graph neural network (gnn) used to tighten the analysis of flow prolongation in fifo multiplexing system. https://github.com/wangweiran0129/Degree_Project_Network_Calculus.

[124] Steffen Bondorf and Jens B Schmitt. The DiscoDNC v2: a comprehensive tool for deterministic network calculus. In *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS '14, pages 44–49, Brussels, BEL, 2014. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 9781631900570. doi: 10.4108/icst.Valuetools.2014.258167. URL https://doi.org/10.4108/icst.Valuetools.2014.258167.

[125] Anne Bouillard, Laurent Jouhet, and Eric Thierry. *Service curves in Network Calculus: dos and don'ts*. Research report, INRIA, 2009. URL https://hal.inria.fr/inria-00431674.

[126] 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)*, volume 4, pages 101–104 vol.4, 2000. doi: 10.1109/ISCAS.2000.858698.

[127] Henrik Schioler, Hans P Schwefel, and Martin B Hansen. Cync: a matlab/simulink toolbox for network calculus. In *Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, ValueTools '07, page 60, Brussels, BEL, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 9789639799004.

[128] Jorn Migge, Marc Fumey, and Marc Boyer. Pegase - a robust and efficient tool for worst-case network traversal time evaluation on afdx. In *Aerospace Technology Conference and Exposition*. SAE International, oct 2011. doi: https://doi.org/10.4271/2011-01-2711. URL https://doi.org/10.4271/2011-01-2711.

[129] Ahlem Mifdaoui and Hamdi Ayed. Wopanets: a tool for worst case performance analysis of embedded networks. In *Computer Aided Modeling, Analysis and Design of Communication Links and Networks (CAMAD), 2010 15th IEEE International Workshop on*, pages 91–95. IEEE, 2010. doi: 10.1109/CAMAD.2010.5686958.

[130] Mark Schmidt, Sebastian Veith, Michael Menth, and Stephan Kehrer. Delaylyzer: A tool for analyzing delay bounds in industrial ethernet networks. In Kai Fischbach and Udo R. Krieger, editors, *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, pages 260–263, Cham, 2014. Springer International Publishing. ISBN 978-3-319-05359-2.

[131] Luca Bisti, Luciano Lenzini, Enzo Mingozzi, and Giovanni Stea. Deborah: A tool for worst-case analysis of fifo tandems. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification, and Validation*, pages 152–168, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-16558-0.

[132] Anne Bouillard and Giovanni Stea. Exact worst-case delay in FIFO-multiplexing feed-forward networks. *IEEE/ACM Transactions on Networking (TON)*, 23(5):1387–1400, 2015. doi: 10.1109/TNET.2014.2332071.

[133] Anne Bouillard and Éric Thierry. Tight performance bounds in the worst-case analysis of feed-forward networks. *Discrete Event Dynamic Systems*, 26(3):383–411, Sep 2016. ISSN 1573-7594. doi: 10.1007/s10626-015-0213-2. URL https://doi.org/10.1007/s10626-015-0213-2.

[134] Anne Bouillard. Stability and performance bounds in cyclic networks using network calculus. In Étienne André and Mariëlle Stoelinga, editors, *Formal Modeling and Analysis of Timed Systems*, pages 96–113, Cham, 2019. Springer International Publishing. ISBN 978-3-030-29662-9.

[135] Sven Kerschbaum, Kai-Steffen Hielscher, and Reinhard German. The need for shaping non-time-critical data in profinet networks. In *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*, pages 160–165, 2016. doi: 10.1109/INDIN.2016.7819151.

[136] Anne Bouillard. Trade-off between accuracy and tractability of network calculus in fifo networks. *Performance Evaluation*, 153:102250, 2022. ISSN 0166-5316. doi: https://doi.org/10.1016/j.peva.2021.102250. URL https://www.sciencedirect.com/science/article/pii/S0166531621000675.

[137] Lucien Rakotomalala, Pierre Roux, and Marc Boyer. Verifying min-plus computations with coq. In *NFM*, volume 12673 of *Lecture Notes in Computer Science*, pages 287–303. Springer, 2021.

[138] IBM ILOG Cplex. V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009.

[139] Michel Berkelaar, Kjell Eikland, and Peter Notebaert. lpsolve, May 2004. URL https://lpsolve.sourceforge.net/5.5/.

[140] Jörg Liebeherr. Duality of the max-plus and min-plus network calculus. *Foundations and Trends in Networking*, 11(3-4):139–282, 2017.

[141] OpenAI. ChatGPT: An ai language model by openai. OpenAI Website, 2023. URL https://openai.com/.

[142] D. B. Chokshi and P. Bhaduri. Modeling fixed priority non-preemptive scheduling with real-time calculus. In *2008 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 387–392, Aug 2008. doi: 10.1109/RTCSA.2008.28.

[143] 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)*, volume 4, pages 101–104 vol.4, May 2000. doi: 10.1109/ISCAS.2000.858698.

[144] Yao-Tzung Wang, Tzung-Pao Lin, and Kuo-Chung Gan. An improved scheduling algorithm for weighted round-robin cell multiplexing in an ATM switch. In *Proceedings of the International Conference on Communications (SUPERCOMM'94)*, pages 1032–1037 vol.2, May 1994. doi: 10.1109/ICC.1994.368945.

[145] Hideyuki Shimonishi, Makiko Yoshida, Ruixue Fan, and Hiroshi Suzuki. An improvement of weighted round robin cell scheduling in ATM networks. In *Proc. of the IEEE Global Telecommunications Conference (GLOBECOM 97)*, volume 2, pages 1119–1123 vol.2, Nov 1997. doi: 10.1109/GLOCOM.1997.638500.

[146] Hemant. M Chaskar and Upamanyu Madhow. Fair scheduling with tunable latency: A round robin approach. In *Proc of the IEEE Global Telecommunications Conference (GLOBECOM'99)*, volume 2, pages 1328–1333. IEEE, 1999.

[147] Dimitrios Stiliadis and Anujan Varma. Latency-rate servers: A general model for analysis of traffic scheduling algorithms. *IEEE/ACM Trans. Netw.*, 6(5):611–624, October 1998. ISSN 1063-6692. doi: 10.1109/90.731196. URL http://dx.doi.org/10.1109/90.731196.

[148] S Nananukul. Latency of weighted round-robin scheduler. *Electronics Letters*, 39(2):256–257, 2003.

[149] Yuming Jiang. Relationship between guaranteed rate server and latency rate server. *Computer Networks*, 43(3):307–315, 2003. ISSN 1389-1286. doi: 10.1016/S1389-1286(03)00276-7. URL http://www.sciencedirect.com/science/article/pii/S1389128603002767.

[150] Jean-Philippe Georges, Thierry Divoux, and Éric Rondeau. Network calculus: application to switched real-time networking. In *Proc. of the 5th Int. ICST Conf. on Performance Evaluation Methodologies and Tools*, VALUETOOLS '11, pages 399–407, ICST, Brussels, Belgium, Belgium, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-1-936968-09-1. URL http://dl.acm.org/citation.cfm?id=2151688.2151733.

[151] Aakash Soni, Xiaoting Li, Jean-Luc Scharbarg, and Christian Fraboul. WCTT analysis of avionics switched ethernet network with WRR scheduling. In *Proc. of the 26th International Conference on Real-Time Networks and Systems (RTNS)*, pages 213–222. ACM, 2018.

[152] Ehsan Mohammadpour, Eleni Stai, and Jean-Yves Le Boudec. Improved delay bound for a service curve element with known transmission rate. *IEEE Networking Letters*, pages 1–1, 2019. doi: 10.1109/LNET.2019.2925176. URL http://infoscience.epfl.ch/record/267840.

[153] D.P. Bertsekas. *Convex Optimization Theory*. Athena Scientific optimization and computation series. Athena Scientific, 2009. ISBN 9781886529311. URL https://books.google.ch/books?id=0H1iQwAACAAJ.

[154] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round-robin. *IEEE/ACM Transactions on Networking*, 4(3):375–385, 1996.

[155] Anna Charny and Jean-Yves Le Boudec. Delay bounds in a network with aggregate scheduling. In *International Workshop on Quality of Future Internet Services*, pages 1–13. Springer, 2000.

[156] Ahlem Mifdaoui and Thierry Leydier. Beyond the Accuracy-Complexity Tradeoffs of CompositionalAnalyses using Network Calculus for Complex Networks. In *10th International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (co-located with RTSS 2017)*, pages pp. 1–8, Paris, France, December 2017. URL https://hal.archives-ouvertes.fr/hal-01690096.

[157] Anne Bouillard. Trade-off between accuracy and tractability of network calculus in fifo networks. *Performance Evaluation*, 153:102250, 2022. ISSN 0166-5316. doi: https://doi.org/10.1016/j.peva.2021.102250. URL https://www.sciencedirect.com/science/article/pii/S0166531621000675.

[158] Aakash Soni and Jean-Luc Scharbarg. Deficit round-robin: Network calculus based worst-case traversal time analysis revisited. In *2022 IEEE 47th Conference on Local Computer Networks (LCN)*, pages 275–278, 2022. doi: 10.1109/LCN53696.2022.9843526.

[159] Marc Boyer and Pierre Roux. A common framework embedding network calculus and event stream theory. working paper or preprint, May 2016. URL https://hal.archives-ouvertes.fr/hal-01311502.

[160] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009. ISBN 0262033844.

[161] A. Bouillard. Stability and performance bounds in cyclic networks using network calculus. In Étienne André and Mariëlle Stoelinga, editors, *Formal Modeling and Analysis of Timed Systems - 17th International Conference, FORMATS 2019, Amsterdam, The Netherlands, August 27-29, 2019, Proceedings*, volume 11750 of *Lecture Notes in Computer Science*, pages 96–113. Springer, 2019. doi: 10.1007/978-3-030-29662-9_6. URL https://doi.org/10.1007/978-3-030-29662-9_6.

[162] Hugo Daigmorte, Marc Boyer, and Luxi Zhao. Modelling in network calculus a TSN architecture mixing Time-Triggered, Credit Based Shaper and Best-Effort queues. working paper or preprint, June 2018. URL https://hal.archives-ouvertes.fr/hal-01814211.

[163] Kai Lampka, Steffen Bondorf, and Jens Schmitt. Achieving efficiency without sacrificing model accuracy: Network calculus on compact domains. In *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 313–318, 2016. doi: 10.1109/MASCOTS.2016.9.

[164] Anne Bouillard and Éric Thierry. An algorithmic toolbox for network calculus. *Discrete Event Dynamic Systems*, 18(1):3–49, 2008. doi: 10.1007/s10626-007-0028-x. URL https://doi.org/10.1007/s10626-007-0028-x.

[165] Rafik Henia, Arne Hamann, Marek Jersak, R. Racu, Kai Richter, and Rolf Ernst. System level performance analysis - the symta/s approach. *Computers and Digital Techniques, IEE Proceedings* -, 152:148–166, 04 2005. doi: 10.1049/ip-cdt:20045088.

[166] Ernesto Wandeler and Lothar Thiele. Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`, 2006. URL http://www.mpa.ethz.ch/Rtctoolbox.

[167] Nan Guan and Wang Yi. Finitary real-time calculus: Efficient performance analysis of distributed embedded systems. In *2013 IEEE 34th Real-Time Systems Symposium*, pages 330–339, 2013. doi: 10.1109/RTSS.2013.40.

[168] Kai Lampka, Steffen Bondorf, Jens B. Schmitt, Nan Guan, and Wang Yi. Generalized finitary real-time calculus. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, 2017. doi: 10.1109/INFOCOM.2017.8056981.

[169] Ehsan Mohammadpour, Eleni Stai, and Jean-Yves Le Boudec. Improved network calculus delay bounds in time-sensitive networks, 2022. URL https://arxiv.org/abs/2204.10906.

[170] P. Massart. The Tight Constant in the Dvoretzky-Kiefer-Wolfowitz Inequality. *The Annals of Probability*, 18(3):1269 – 1283, 1990. doi: 10.1214/aop/1176990746. URL https://doi.org/10.1214/aop/1176990746.

[171] Hugo Daigmorte and Marc Boyer. Traversal time for weakly synchronized can bus. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, page 35–44, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450347877. doi: 10.1145/2997465.2997477. URL https://doi.org/10.1145/2997465.2997477.

[172] Yuming Jiang. A basic stochastic network calculus. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '06, page 123–134, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933085. doi: 10.1145/1159913.1159929. URL https://doi.org/10.1145/1159913.1159929.

[173] Florin Ciucu, Almut Burchard, and Jörg Liebeherr. Scaling properties of statistical end-to-end bounds in the network calculus. *IEEE/ACM Transactions on Networking (ToN)*, 14(6):2300–2312, 2006.

[174] J.E. Gentle. *Computational Statistics*. Statistics and Computing. Springer New York, 2009. ISBN 9780387981444. URL https://books.google.ch/books?id=mQ5KAAAAQBAJ.

[175] Anne Bouillard, Nadir Farhi, and Bruno Gaujal. Packetization and packet curves in network calculus. In *Performance Evaluation Methodologies and Tools (VALUETOOLS), 2012 6th International Conference on*, pages 136–137. IEEE, 2012.

[176] Steffen Bondorf, Paul Nikolaus, and Jens B. Schmitt. Quality and cost of deterministic network calculus: Design and evaluation of an accurate and fast analysis. In *SIGMETRICS (Abstracts)*, page 65. ACM, 2017.

[177] Anne Bouillard, Laurent Jouhet, and Eric Thierry. Tight performance bounds in the worst-case analysis of feed-forward networks. In *INFOCOM*, pages 1316–1324. IEEE, 2010.

List of Publications

Here is the list of my publications as a PhD student at EPFL.

1. **Seyed Mohammadhossein Tabatabaee**, Jean-Yves Le Boudec, and Marc Boyer. Interleaved weighted round-robin: A network calculus analysis. In *2020 32nd International Teletraffic Congress (ITC 32)*, pages 64–72, 2020. doi: 10.1109/ITC3249928.2020.000162.
2. **Seyed Mohammadhossein Tabatabaee**, Jean-Yves Le Boudec, and Marc Boyer. Interleaved weighted round-robin: A network calculus analysis.*IEICE Transactions on Communications*, E104.B(12):1479–1493, 2021. doi: 10.1587/transcom.2021ITI00013.
3. **Seyed Mohammadhossein Tabatabaee** and Jean-Yves Le Boudec. Deficit round-robin: A second network calculus analysis. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 171–183, 2021. doi: 10.1109/RTAS52030.2021.000224.
4. **Seyed Mohammadhossein Tabatabaee** and Jean-Yves Le Boudec. Deficit round-robin: A second network calculus analysis. *IEEE/ACM Transactions on Networking*, pages 1–15, 2022. doi: 10.1109/TNET.2022.31647725.
5. **Seyed Mohammadhossein Tabatabaee**, Anne Bouillard, and Jean-Yves Le Boudec. Worst-case delay analysis of time-sensitive networks with deficit round-robin, 2022. URL https://arxiv.org/abs/2208.114006.
6. **Seyed Mohammadhossein Tabatabaee**,Marc Boyer, Jean-Yves Le Boudec, and Jörn Migge. Efficient and Accurate Handling of Periodic Flows in Time-Sensitive Networks. In *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. URL-http://infoscience.epfl.ch/record/302640.
7. **Seyed Mohammadhossein Tabatabaee**, Anne Bouillard, and Jean-Yves Le Boudec. Quasi-deterministic burstiness bound for aggregate of independent, periodic flows. Accepted at *Quantitative Evaluation of Systems (QEST) 2023*. URL https: //arxiv.org/abs/2305.149467.
8. Chun-Tso Tsai, **Seyed Mohammadhossein Tabatabaee**, Stéphan Plassart, and Jean-Yves Le Boudec. Saihu: A common interface of worst-case delay analysis tools for time-sensitive networks, 2023. URL https://arxiv.org/abs/2303.14565.

Seyed Mohammadhossein **TABATABAEE**

Curriculum Vitae

✉ *smh.tabatabaee96@gmail.com*
www.linkedin.com/in/hosseintabatabaee/

Education

2019–2023 **PhD in Computer and Communication Science**, *École Polytechnique Fédérale de Lausanne (EPFL)*, Lausanne, Switzerland.

2015–1019 **B.Sc in Electrical Engineering**, *Sharif University of Technology*, Tehran, Iran.
Major Telecommunication

Core Experience

2019–2023 **Research Assistant**, *Laboratory for Computer Communications and Applications (EPFL)*.
- Worst-case analysis of large-scale time-sensitive networks, as in the context of IEEE TSN and IETF DetNet, in terms of delay, delay-jitter, and buffer bounds.
- Formal worst-case service modeling of wide-spread schedulers such as Deficit Round-Robin (DRR) and Interleaved Weighted Round-Robin (IWRR).
- Formal, large scale algorithm for end-to-end delay bounds computation for networks of generic shapes, using a distributed, parallel computing model with shared memory.
- Implementation and optimization of Linear Programs (LP) and Mixed-Integer Linear Programs (MILP) for delay bound computation.
- See Google Scholar for the list of publications.

Summer 2022 **Research Intern**, *Real-Time-at-Work (RTaW)*, Grenoble, France.
- RTaW is leading the way in Ethernet Time-Sensitive Networking (TSN) design, performance evaluation and automated configuration tools.
- Development and validation of an algorithm that provides tighter delay bounds for networks of generic topology at considerably less complexity.
- Implemented in Java and integrated in RTaW's existing tool.

Additional Experience

2022–2023 **Supervising a Semester Project**, *EPFL*.
- Saihu : A common, language-independent interface for worst-case delay analysis of time-sensitive networks.
- Fully integration of the most frequently analysis tools, including NetCal/DNC (in Java), xTFA (in Python), and panco (in Python).

2021–2022 **Supervising a Master Thesis**, *EPFL*.
- Implementation a Graph Neural Network (GNN) that is used to tighten the delay bounds.
- Applied Fast Gradient Signed Method (FGSM) adversarial attack to evaluate its robustness.
- Implementation in Python.

2021–2022 **Supervising a Semester Project**, *EPFL*.
- Formally verified the DRR modeling and algorithm, using Coq proof assistant.
- Formally verified findings of our papers, using Coq proof assistant.

Summer 2018 **Research Intern**, *Institute of Network Coding*, CUHK, Hong Kong.
- Study of Lattice-based Public-key Cryptography.

2018 **Summer School**, *The Cornell, Maryland, Max Planck Pre-doctoral Research Summer School*, Saarbrücken, Germany.

2017–2018 **Research Intern**, *Brain Engineering Research Center at IPM*, Tehran, Iran.
- Analyzed the effect of Prediction in a Non-random fMRI Experiment.
- Applied regression model different machine learning approaches.

2017 - 2019 **Other Projects**, *Sharif University of Technology*, Tehran, Iran.

- Designed a wireless communication system based on the IEEE 802.11n standard in MATLAB/Simulink.
- Implementation of a server-based peer-to-peer chat application using TCP protocol in Python.
- Implementation and simulation physical and MAC layers of a network protocol stack in NS3.
- EEG Signal Processing using Machine Learning to help people who cannot talk in MATLAB.
- Implementation of a distributed admission control using Software-defined Networking (SDN) in MATLAB.
- Designed and Implementation of BJT Audio Amplifier and AM Transmitter.
- Designed and Implementation of an elevator controller on a FPGA.

Expertise

Technical

Programming Matlab, Java, Python, C++, Past:{Scala, VHDL, Verilog}.

Tools Wireshark, LaTex, SVN, Git, Past:{Hspice, Altium Designer, Pspice, AutoCAD}

OS MAC OS, Linux, Windows.

Concepts

Networking TCP/IP, Congestion Control, ARP, Quic, Bier Rouing, OSPF, BGP, MPLS, VPN, TSN, DetNet.

Communication IEEE 802.11, 1G/2G/3G, OFDM, FDM, TDMA, CDMA, Cellular Networks, Encoding, Decoding.

Security Symmetric/Asymmetric encryption, Side-channel attacks, SQL injection.

ML Regression, Classification, Over/Underfitting, Regularization.

Soft Skills

Critical and structural thinking, Team-working, Stress and time management, Project management.

Related Teaching Experience

2020-2023 **TCP/IP Networking**, *Teaching Assistant*, EPFL.

- Lecture on Congestion Control and BGP .
- Responsible for labs with hands-on exercises on socket programming, TCP congestion control, IPv4/IPv6 interworking, OSPF, BGP, DNS, TCP, UDP, Https, TLS, tunneling, routing, and network security.

2022 **Smart Grids Technology**, *Teaching Assistant*, EPFL.

- Lecture and labs with hands-on exercises on Introduction to TCP/IP .

2021 **Performance Evaluation**, *Teaching Assistant*, EPFL.

Responsible for the lab problems that involved performance patterns (bottlenecks, congestion collapse), model fitting and forecasting, discrete-event simulation and queuing theory.

2020 **Advanced information, computation, communication II**, *Teaching Assistant*, EPFL.

Responsible for the homework problems that involved probability distribution, channel encoding/decoding, and cryptography.

Honors and Awards

2022 **Best Presentation Award** , *6th Workshop on Network Calculus (WoNeCa-6)*.

2021 **Outstanding Performance Award**, *EPFL*.

2021 **Teaching Assistant Award**, *EPFL*.

2020 **Best Presentation Award** , *5th Workshop on Network Calculus (WoNeCa-5)*.

2014 **Silver Medal**, *National Astronomy and Astrophysics Olympiad*, Iran.

Languages

English (C1, fluent), French (A2/B1, intermediate), German (A1, elementary), Farsi (native).

Extra-curricular Activities

Swimming, Road trips, Hiking, and Traveling.

Personal Information

194 Married, Swiss driver's license (Type B).