Worst-case Delay Analysis of Time-Sensitive Networks with Deficit Round-Robin

Seyed Mohammadhossein Tabatabaee
EPFL
Lausanne, Switzerland
hossein.tabatabaee@epfl.ch

Anne Bouillard
Huawei Technologies
Paris, France
anne.bouillard@huawei.com

Jean-Yves Le Boudec
EPFL
Lausanne, Switzerland
jean-yves.leboudec@epfl.ch

***Abstract*—In feed-forward time-sensitive networks with Deficit Round-Robin (DRR), worst-case delay bounds were obtained by combining Total Flow Analysis (TFA) with the strict service curve characterization of DRR by Tabatabaee et al. The latter is the best-known single server analysis of DRR, however the former is dominated by Polynomial-size Linear Programming (PLP), which improves the TFA bounds and stability region, but was never applied to DRR networks. 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. Second, we extend the methodology to support networks with cyclic dependencies: This raises further dependency loops, as, on one hand, DRR strict service curves rely on traffic characteristics inside the network, which comes as output of the network analysis, and on the other hand, TFA or PLP requires prior knowledge of the DRR service curves. This can be solved by iterative methods, however PLP itself requires making cuts, which imposes other levels of iteration, and it is not clear how to combine them. We propose a generic method, called PLP-DRR, for combining all the iterations sequentially or in parallel. We show that the obtained bounds are always valid even before convergence; furthermore, at convergence, the bounds are the same regardless of how the iterations are combined. This provides the best-known worst-case bounds for time-sensitive networks, with general topology, with DRR. We apply the method to an industrial network, where we find significant improvements compared to the state-of-the-art.**

I. Introduction

Deficit Round-Robin (DRR) [1] is a scheduling algorithm that is often used in real-time systems or communication networks for scheduling tasks, or packets. With DRR, every queue is assigned a *quantum* that is a static number. Queues are visited in *round-robin* manner; the service received at every visit, which is measured in bits for communication networks or in seconds for task processing systems, is up to the quantum value. Tasks or packets have variable sizes and it might happen that, during a visit of the server, at least one task or packet, which cannot be served, remains in the queue; this is 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 is efficient (when a queue is idle, the server capacity is available to other queues). As it has low complexity and very efficient implementations exist [2], it is widely used.

DRR can be applied to time-sensitive networks where bounds on worst-case delays (not on average) are required. Here, flows with similar delay requirements can be mapped to the same class, every class corresponds to one DRR queue at every node, and packets inside one class are handled first in first out (FIFO). Also, flows are limited at sources by arrival curve constraints, i.e., limits to the number of bits that can be sent over any time interval. Finding 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 a recent RTAS conference, the authors in [3] derive a strict service curve for DRR, i.e., a function that lower bounds the amount of service received by every DRR queue. Delay bounds are then derived by using network calculus formulas. This method captures the interference of competing DRR classes and, as of today, provides the best known worst-case delay bounds [4]–[8]. We call this method the *DRR strict service curve.*

For the latter step, per-class network analysis, Total Flow Analysis (TFA) [9] was used in [3]. 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. The authors in [3] solve this problem by considering only feed-forward networks (in the application example, they constrain flow routes to avoid cyclic dependencies). However, cyclic dependencies are frequent in time-sensitive networks and cannot be ignored. Recent versions of TFA [10], [11] apply to networks with cyclic dependencies and can therefore be used: as a side-contribution, in Section VI-A, we show how to apply TFA to DRR networks with cyclic dependencies, by developing, and proving the validity of, an iterative procedure.

However, our main contribution goes well beyond TFA; indeed, it is known that TFA is outperformed by polynomial-size linear programming (PLP) [12], which 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 [13] and flow prolongations [14] also tend to dominate TFA, however unlike PLP, they do not apply to generic topologies. This motivates the goal of this paper, 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 in 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; 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; 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 analyse DRR networks requires to introduce 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 depend 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, 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 4). Still, some concrete implementations of the method may have better execution times, and we propose two such concrete, parallel implementation methods, which we apply to the industrial network in [3].

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 1, we extend the PLP methodology to obtain such per-aggregate bounds. Second, PLP requires convex service curves; [3] provides 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 V-B).

The contributions of this paper 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 the ones obtained with TFA.
- We develop, and show the validity of, a method 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 apply them to the industrial network in [3]. We find that the delay bounds are significantly better than the state of the art.

The rest of the paper is organized as follows. Section II describes the system model, including DRR operation, the network under study and the resulting graphs. Section III gives the necessary background on DRR strict service curve, TFA and PLP. Section IV gives 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 V. The details of PLP-DRR are described in Section VI, including statements about the validity and the uniqueness of the obtained bounds. Section VII contains proofs of theorems. Section VIII applies the method to the industrial network in [3] and illustrates the obtained improvements on delay bounds. Section IX concludes the paper.

II. System Model

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

A. Deficit Round-Robin Scheduling

A DRR subsystem serves n inputs, has one queue per input. Each queue c is assigned a quantum Q_c. DRR runs an infinite loop of *rounds*. In one round, if queue c is non-empty, a service for this queue starts and its *deficit* is increased by Q_c. The service ends when either the deficit is smaller than the size of the head-of-the-line packet or the queue becomes empty. In the latter case, the deficit is set to zero. Packet sending duration depends both on the packet size and the amount of service available for the entire DRR subsystem.

The DRR subsystem is placed in a larger system and can compete with other queuing subsystems. Then, the service offered to the DRR subsystem might not be instantly available. This can be modelled by means of a strict service curve $B()$, i.e., a function such at least $B(\tau)$ bits of any DRR class are served during any period of time of duration τ where the DRR subsystem is backlogged. A frequently used strict service curve is the rate-latency function with rate R and latency T, defined by $\beta_{R,T}(t) = R[t-T]^+$, where we use the notation $[x]^+ = \max\{x, 0\}$. For example, when the DRR subsystem is at the highest priority on a non-preemptive server with line rate R, this gives a rate-latency strict service curve with rate R and latency $\frac{R}{L^{\max}}$ 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 modelled with a more complex strict service curve [9, Section 8.3.2].

Here we use the language of communication networks, yet 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" [15], [16].

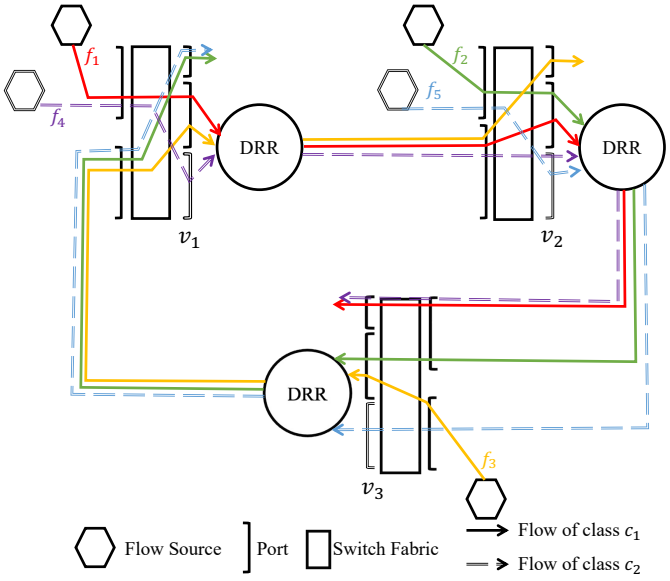
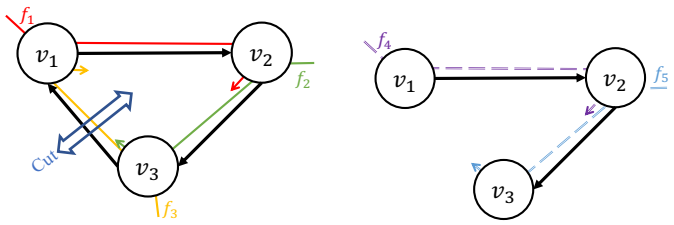


Fig. 1: Toy Network with 2 DRR classes. Flows f_1, f_2, and f_3 belong to class c_1; flow f_4 and f_5 belong to class c_2.

B. Network Model and Resulting Graphs

1) Device Model: Devices represent switches or routers and consist of input ports, output ports, and a switching fabric.



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

Fig. 2: The graphs induced by flows of class c_1 and c_2 of toy network of Fig. 1, also showing the flows path. $\mathcal{G}_{c_1}$ has one cyclic dependency, $\mathcal{G}_{c_2}$ has none.

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 II-A. See Fig. 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).

2) Flow Model: We assume that there are n classes of traffic $1, \ldots, n$ in the system, and flows are statically assigned a class and a path. Each flow f is constrained at source by a token-bucket arrival curve defined by $\gamma_{r_f,b_f}(t) = r_f t + b_f$ for $t > 0$ and $\gamma_{r_f,b_f}(t) = 0$ for $t = 0$. Having a token-bucket arrival curve with a rate r_f and a burst b_f means that the number of bits observed for this flow in any time interval of duration τ cannot be more than $b_f + r_f\tau$ bits.

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 [17]. 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. 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. 2. Graph $\mathcal{G}_{c_1}$ has a cycle; the figure shows one possible artificial cut to create a tree.

TABLE I: Notation List

$\mathcal{V}, v$	The set of all output ports, an output port
$\mathcal{G}_c = (\mathcal{V}_c, \mathcal{E}_c)$	The graph induced by flows of class c
$\mathcal{V}_c$	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
E_c^{cut}	Cutset: removing E_c^{cut} creates a tree or a forest for class c
B^v	Aggregate strict service curve offered to v
f, $\alpha_f = \gamma_{r_f, b_f}$	A flow, its token-bucket arrival curve
$\text{In}_c(v) \subset \mathcal{E}_c$	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
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
z_c^{cut}	Upper bounds on the burstiness of flows of class c at cuts E_c^{cut}
z^{cut}	Collection of z_c^{cut} for all classes c
d_c^v	Delay bound on node v for class c
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^v	Strict service curve offered to class c at node v
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)
b_c^v	bound on aggregate burstiness of flows of class c that exits node v
b_c^e	bound on aggregate burstiness of flows of class c carried by edge e
b	Collection of b_c^v and b_c^e for every class c, every edge e, and every node v
$\beta_{R,T}$	$\beta_{R,T}(t) = R[t-T]^+$, 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

III. Network Calculus Background

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 III-A; 2) the analysis of one FIFO network per DRR class. Two methods are presented, TFA in Section III-B and PLP in Section III-C.

A. Strict Service Curves of DRR

Authors in [18] (extended version of [3]) find DRR strict service curves for degraded operational mode (i.e., where no assumption on the arrival curve of the interfering traffic is assumed) and for non-degraded operational mode (i.e., where some arrival curves can be assumed for the interfering traffic). They show that their resulting delay bounds dominate all previous works for single node analysis [4]–[8], and hence, are the best, proven ones.

1) Degraded Operational Mode: Let v be a node shared by n classes that uses DRR, as explained in Section II-A, 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 $\beta_c^{\text{CDM},v}$ 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 A-A for more details.

Authors in [18] show that 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 $\beta_c^{\text{nc},v}$, as it corresponds to the first service round. We have $\beta_c^{\text{nc},v}(t) = \min\left(\beta_{R^v,T_1}(t), q_c^v\right)$ where T_1 is the maximum period of time during which no data of class c can be served and q_c^v corresponds to the minimum amount of data guaranteed for class c during each round (see Fig. 3); the exact values are given in Appendix A-A. We use it as follows in Section V-B: since $\beta_c^{\text{nc},v}$ is a strict service curve, we can replace any other strict service curve for class c, β_c^v, by $\max(\beta_c^v, \beta_c^{\text{nc},v})$, which is also a strict service curve.

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. Authors in [18] presents a method that starts from service curves with no assumption on the interfering traffic (i.e., $\beta_c^{\text{CDM},v}$ explained in Section III-A1), and iteratively improves them by taking

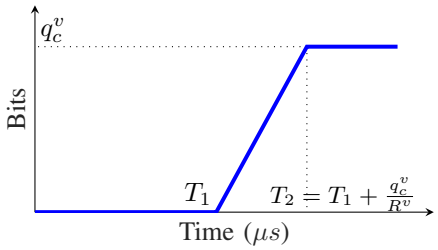


Fig. 3: A non-convex part of a service curve $\beta^{\text{nc},v}(t) = \min\left(\beta_{R^v,T_1}(t), q_c^v\right)$.

into account the arrival curve constraints of interfering traffic. We call $\text{DRRservice}_v^{\text{inputArrival}}\left(\alpha^v\right)$ 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 A-B for more details.).

We also use Lemma 2 in [18], 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 $\text{DRRservice}_v^{\text{outputBurst}}\left(\beta^v, b^v\right)$ the function that implements Lemma 2 in [18] and returns an improved collection of strict service curves for all classes at node v (See (12) in Appendix A-B for details). In the above, β^v and b^v are the collection of β_c^v strict service curves and 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, the obtained strict service curves are the maximum of a finite number of rate-latency functions: the latency of such functions is a linear function of output burstinesses b_c^v, and the rates depend only on arrival rates r_c.

B. Total Flow Analysis (TFA)

Total Flow Analysis (TFA) [11], [19], [20] 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. 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 [11]. 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, FPTFA, SyncTFA, AsyncTFA, and AltTFA) are equivalent, i.e., they give the same bounds and stability regions [11]. 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.

C. Polynomial-size Linear Programming (PLP)

Polynomial-size linear program (PLP) computes end-to-end delay bounds in FIFO networks [12], 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 the linear programs is straightforward for tree topologies. The analysis of general topologies requires to first make 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:

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 V. 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.

A detailed background on these linear programs is presented in Appendix B.

IV. Overview of the Proposed Method: PLP-DRR

Our method, called "PLP-DRR", applies the PLP methodology to DRR and is illustrated in Fig. 4. The starting point is the collection of per-class graphs and a cutset. We use the following notations:

- $\beta = (\beta_c^v)_{1\le c\le n, v\in\mathcal{V}}$, is a valid collection of strict service curves of each class c at each node v,
- $d = (d_c^v)_{1\le c\le n, v\in\mathcal{V}}$, 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,
- $b = (b_c^g)_{1\le c\le n, g\in\mathcal{V}\cup\mathcal{E}}$ 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.

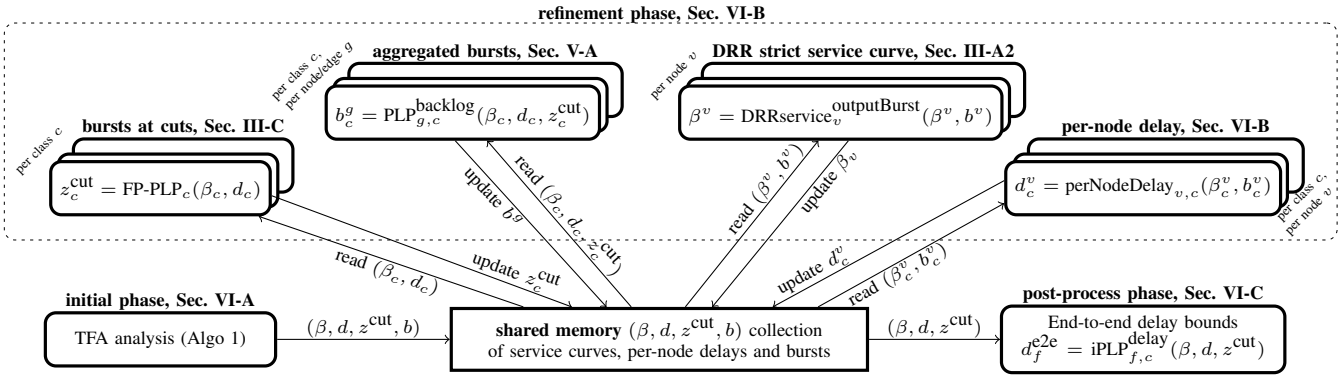


Fig. 4: Overview of the method. The refinement phase consists in applying, in any order, any of the four types of refinement blocks shown on the top of the figure, which each improve the bounds stored in the shared memory. The refinement phase may be stopped using any criterion, such as convergence of 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.

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 $\text{PLP}^{\text{backlog}}$, which, in turn, allows to compute better service curves β using the DRR service curve method recalled in Section III-A. The linear program in $\text{PLP}^{\text{backlog}}$ uses per-node delay bounds d, which can in turn be improved by re-running TFA whenever β is improved, and then $\text{PLP}^{\text{backlog}}$ could also be re-run. Also, better β and d allows 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. 4). As we show in Section VI-B, 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 $\text{PLP}^{\text{backlog}}$ applied to all flows in the aggregate. We improve both the obtained bound and the computing time by using the modification of $\text{PLP}^{\text{backlog}}$ described in Section V-A.

The third phase of PLP is to obtain end-to-end bounds by applying one instance of $\text{PLP}^{\text{delay}}$ to every flow of interest. Here, we use $\text{PLP}^{\text{delay}}$ with one improvement (iPLP), which allows to use non-convex service curves at the expense of adding a few binary variables to the linear program (Section V-B). Such an improvement could also be used in the refinement phase, but we found experimentally that this would have no noticeable effect.

V. 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 $\text{PLP}_{f,i}^{\text{backlog}}$ and $\text{PLP}_{f,i}^{\text{delay}}$ in step 3) of PLP as explained in Section III-C; 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 [12, Section 4]. 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 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 $k \in \{0, \dots, \text{dp}(v)\}$. Map to $\mathbf{t}_{(j,k)}$ of [12] by mapping j to v.

Process Variables: For every node v and v' and every f at v, define $\mathbf{Ft}^{v,f}_{v',k}$ with $k \in \{0,\dots,\mathrm{dp}(v')\}$, 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 $\mathbf{t}_{(v',k)}$. It can be mapped to $\mathbf{F}^j_i\mathbf{t}_{(j',k)}$ of [12] by mapping j to v, i to f, and j' to v'. In the next paragraphs, we only presents the parts of the linear program that are modified from [12]. The complete linear programs are presented in Appendix B.

A. 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: $\forall f \in F,\ \forall k,\ k \in [0,\ \mathrm{dp}(v_f)],\ \mathbf{Ft}^{v_f,f}_{v_0,0} - \mathbf{Ft}^{v_f,f}_{v_f,k} \le b_f + r_f(\mathbf{t}_{(v_0,0)} - \mathbf{t}_{(v_f,k)})$;
- 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 $\mathrm{PLP}^{\mathrm{backlog}}_{v,c}$ 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 $\mathrm{DRRservice}^{\mathrm{outputBurst}}_{v}$ in the refinement phase to compute DRR strict service curves. We also apply this program when e is an edge, and also call it $\mathrm{PLP}^{\mathrm{backlog}}_{e,c}$. 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 $\mathrm{perNodeDelay}_{v,c}$ to compute per-node delay bounds. Theorem 1 is proved in Section VII-A.

Theorem 1 (PLP to Upper-bound the Aggregate Burstiness of Flows)**.** *The solution of $PLP^{backlog}_{g,c}$ is a valid bound on aggregate burstiness of flows carried by edge or node g.*

B. 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 $\max(\beta^v_c, \beta^{nc,v}_c)$, where β^v_c is piece-wise linear convex (i.e., $\beta^v_c = \max_p \beta_{R^v_p,T^v_p}$)), and $\beta^{\mathrm{nc},v}_c = \min(\beta_{R^v,T_1}, q^v_c)$ as described in Section III-A. 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 $\mathbf{At}^v_u = \sum_{f\in\mathrm{In}(v)} \mathbf{Ft}^{u,f}_{u,\mathrm{dp}(u)}$ and $\mathbf{At}^v_v = \sum_{f\in\mathrm{In}(v)} \mathbf{Ft}^{v,f}_{v,\mathrm{dp}(v)}$, where $u = \mathrm{sc}(v)$.

The original service curve constraints of PLP are kept:

$$\mathbf{At}^v_u - \mathbf{At}^v_v \ge 0; \quad (1)$$

$$\forall p,\ \mathbf{At}^v_u - \mathbf{At}^v_v \ge R^v_p\left(\mathbf{t}_{(u,\mathrm{dp}(u))} - \mathbf{t}_{(v,\mathrm{dp}(v))} - T^v_p\right). \quad (2)$$

Define $\mathbf{b}_v \in \{0,1\}$ and consider a large enough positive M, and add the following constraints:

$$\mathbf{At}^v_u - \mathbf{At}^v_v \ge R^v(\mathbf{t}_{(u,\mathrm{dp}(u))} - \mathbf{t}_{(v,\mathrm{dp}(v))} - T_1) - M\mathbf{b}_v; \quad (3)$$

$$\mathbf{At}^v_u - \mathbf{At}^v_v \le q^v_c + M\mathbf{b}_v; \quad (4)$$

$$\mathbf{At}^v_u - \mathbf{At}^v_v \ge q^v_c - M(1-\mathbf{b}_v). \quad (5)$$

We let $\mathrm{iPLP}^{\mathrm{delay}}_{f,c}$ denote this Integer, Polynomial-sized Linear Program. Theorem 2 is proved in Section VII-B.

Theorem 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.*

VI. Our Proposed Method: PLP-DRR

In this section, we provide the details of our generic method and we present two concrete implementations.

A. Initial phase: TFA analysis

The goal of the first phase is to provide valid values for $(\beta, d, z^{\mathrm{cut}}, b)$, using TFA. Specifically, we want to analyze the original (uncut) network using TFA (note that TFA itself does not necessarily require cuts [11]). In networks with cyclic dependencies, the TFA analysis of a DRR system of [18] 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 $\beta^{\mathrm{CDM},v}_c$ 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 III-A1. 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 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 III-B) 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 a 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.

Algorithm 1: Initial Phase: TFA analysis

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** $(d_c, z_c) \leftarrow \text{GenericTFA}_c\,(\beta_c)$;
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** $(\beta, d, z^{\text{cut}}, b)$

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 next iteration, more stable classes might be obtained and so on.

Theorem 3 (Correctness and Convergence of TFA Analysis with DRR)**.** *Consider a networks with DRR scheduling per class, as described in Section II, and consider Algorithm 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 VII-C. At this point we have obtained a value of $(\beta, d, z^{\text{cut}}, b)$ that constitute valid bounds.

B. 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 IV, 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 III-C);
- $b_c^g \leftarrow \text{PLP}_{g,c}^{\text{backlog}}(\beta_c, d_c, z_c^{\text{cut}})$: 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 V-A);
- $\beta^v \leftarrow \text{DRRservice}_v^{\text{outputBurst}}\,(\beta^v, b^v)$: computes the DRR strict service curves at node v given the output burstiness bounds at this node (Section III-A).
- $d_c^v \leftarrow \text{perNodeDelay}_{v,c}(\beta_c^v, b_c)$: 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 [12], [20], [21]. Then, we take into account the effect of line shaping and of the packetizer as in Section VI-B of [10].

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.

1) Generic Scheme: A Distributed Computing Model with Shared Memory: The generic scheme is presented in Fig. 4. 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 VI-A. 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 $\text{FP-PLP}_c()$ for some class c, or $\text{PLP}_{g,c}^{\text{backlog}}()$ 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 $\text{FP-PLP}_c()$, 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 $\text{FP-PLP}_c()$, the worker computes $y = z_c^{\text{cut}}$.
- 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 componentwise 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.

Algorithm 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** $(\beta, d, z^{\text{cut}}, b)$

Theorem 4 (Correctness and Convergence of PLP Refinement Phase of Section VI-B1)**.** *Consider a networks with DRR scheduling per class, as described in Section II, 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.*
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.*

2) Two Implementations of PLP Refinements: We presented the generic presentation of this phase. By Theorem 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 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 that the operation on its predecessors are 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, that can be computed in parallel (lines 6-7).

The two implementations are illustrated in Fig. 5. 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 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). 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.

C. Post-process Phase: Computing the End-to-end Delay

When the refinement phase of Section VI-B 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 III-A, delays can be improved when considering the non-convex strict service curve described in Fig. 3. Thus, we apply $\text{iPLP}^{\text{delay}}$ to compute end-to-end delay bounds.

VII. Proofs

A. Proof of Theorem 1

We first prove Lemma 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 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) $\mathcal{B}$ 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 $A - D \leq \mathcal{B}$.

Lemma 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* $\mathcal{B}$ *where* $r = \sum_{f \in F} r_f$*, i.e.,* $\gamma_{r,\mathcal{B}}$.

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 [12], and both theorems give exactly the same result. Also, this result already appears in Corollary 2 of [22], 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 $D(t) - D(s) \leq \gamma_{r,\mathcal{B}}(t - s) = \mathcal{B} + r(t - s)$.

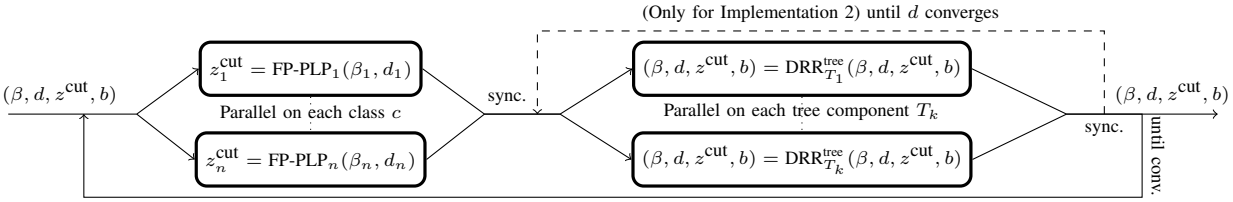


Fig. 5: Two implementations of the refinement phases with parallelization and shared memory. Implementation 1 (without the dashed arrow): alternating between the FP-PLP and $\text{DRR}^{\text{tree}}_T$ blocks; Implementation 2 (with the dashed arrow): convergence of $\text{DRR}^{\text{tree}}_T$ before the execution of FP-PLP.

We first prove that $A(s) - D(s) + H \leq \mathcal{B}$ with $H = b - \bar{b}(s)$ and $\bar{b}(s) \stackrel{def}{=} \sup_{u \leq s}\{A(s) - A(u) - r(s-u)\}$. 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 $A'(s^+) - D'(s^+) \leq \mathcal{B}$ 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 $A(s) - D(s) + H \leq \mathcal{B}$.

To conclude, we notice that $D(t) \stackrel{(A1)}{\leq} A(t) \leq A'(t) = A(s) + H + r(t-s)$, so $D(t) - D(s) \leq A(s) + H + r(t-s) - D(s) \leq \mathcal{B} + r(t-s)$. □

Theorem 5 of [12] 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 [12] 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, $\mathbf{Ft}^{v_f,f}_{v_0,0}$ (resp. $\mathbf{Ft}^{v_0,f}_{v_0,0}$) 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 $\mathbf{t}_{(v_0,0)}$: $\sum_{f \in F} \mathbf{Ft}^{v_f,f}_{v_0,0} - \mathbf{Ft}^{v_0,f}_{v_0,0}$ is a backlog bound the aggregate flows and $\text{PLP}^{\text{backlog}}_{g,c}$ 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 1 this is a bound on the output burstiness. □

B. Proof of Theorem 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 satisfy (with the notations previously introduced) $\mathbf{At}^v_u - \mathbf{At}^v_v \geq \max(\beta^v_c, \beta^{nc,v}_c)(\mathbf{t}_{(u,\text{dp}(u))} - \mathbf{t}_{(v,\text{dp}(v))})$. From (1) and (2), $\mathbf{At}^v_u - \mathbf{At}^v_v \geq \beta^v_c(\mathbf{t}_{(u,\text{dp}(u))} - \mathbf{t}_{(v,\text{dp}(v))})$ holds, so we just need to focus on $\beta^{nc,v}_c$, and distinguish two cases, depending on the value of $\mathbf{b}_v$: if $\mathbf{b}_v = 1$, then (3) and (4) are dummy (because M is large enough) and from (5), $\mathbf{At}^v_u - \mathbf{At}^v_v \geq q^v_c \geq \beta^{nc,v}_c(\mathbf{t}_{(u,\text{dp}(u))} - \mathbf{t}_{(v,\text{dp}(v))})$. Otherwise, $\mathbf{b}_v = 0$ and the (non dummy) constraints from (3) and (4) enforce that $\mathbf{At}^v_u - \mathbf{At}^v_v \leq q^v_c$ and under this constraint, $\mathbf{At}^v_u - \mathbf{At}^v_v \geq R^v(\mathbf{t}_{(u,\text{dp}(u))} - \mathbf{t}_{(v,\text{dp}(v))} - T_1) \geq \beta^{nc,v}_c(\mathbf{t}_{(u,\text{dp}(u))} - \mathbf{t}_{(v,\text{dp}(v))})$. 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). □

C. Proof of Theorems 3 and 4

We use the following lemma. We assume a finite set of isotone mappings $\mathcal{H}$, in $\mathcal{C} \to \mathcal{C}$ with $\mathcal{C} \subseteq (\mathbb{R}^+ \cup \{+\infty\})^I$. An *execution* of $\mathcal{H}$ is $(h_k, s_k, t_k, u_k)_{k \geq 1}$ such that: $\forall k \geq 1$, (C1) $h_k \in \mathcal{H}$; (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 $h \in \mathcal{H}$ is executed infinitely many times; (C6) $\lim_{k \to \infty} s_k = \infty$. In the description of Section VI-B, 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 $x_0 \in \mathcal{C}$ 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 2. *There exists x^* such that for all executions of $\mathcal{H}$ 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 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 II-A, 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 $x = (T, d, z^{\text{cut}}, b)$.

1) Validity: Assuming valid initial bounds, the four types of functions used, PLP$^{\text{backlog}}$, FP-PLP, DRRservice$^{\text{outputBurst}}$ and perNodeDelay (defining set $\mathcal{H}$), 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.

2) Convergence: We then set $\mathcal{C}$ as the set of valid parameters for the problem and apply Lemma 2 where s_k, t_k and u_k respectively correspond to the reading time, locking and unlocking times of the write operation. (C1)–C(3) 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 3. Consider lines 2-6 of Algorithm 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 two latencies per node and class (from line 1). Two types of functions are used: GenericTFA and DRRservice$^{\text{inputArrival}}$ (defining set $\mathcal{H}$). 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 2 can be applied: $x(t)$ converges in $\mathbb{R}^+ \cup \{+\infty\}$. □

VIII. Numerical Evaluation

We use the network of Fig. 7, a test configuration provided by Airbus in [21]. The industrial-sized case study of [3] is based on this network in [23]. It includes 96 end-systems, 8 switches, 984 flows, and 6412 possible paths. The rate of the links are equal to $R = 1$ Gb/s, and every switch S_i has a switching 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. At every DRR scheduler, the quanta are 3070 bytes for the critical class, 1535 bytes for the multimedia and best-effort classes. 128 multicast flows, with 834 destinations, are critical; they have a maximum packet-size of 150 bytes. 500 multicast flows, with 3845 destinations, are multimedia; they have a maximum packet-size of 500 bytes. 266 multicast flows, with 1733 destinations, are best-effort; they have a maximum packet-size of 1535 bytes. For every flow, the path from the source to a destination can traverse at most 4 switches. In [3], as their method only apply to feed-forward networks, flows paths are chosen randomly with the constraint that graphs induced by flows are feed-forward. In this paper, 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.

As of today, there is no other result that computes bounds on worst-case delay of DRR networks with cyclic dependencies. 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: we apply initial phase only; this is the best that can be obtained with TFA and DRR service curves.

2) TFA + 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 V-A 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. 6 (a),(b) 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. 6 (c), we compare 3) to our full method.

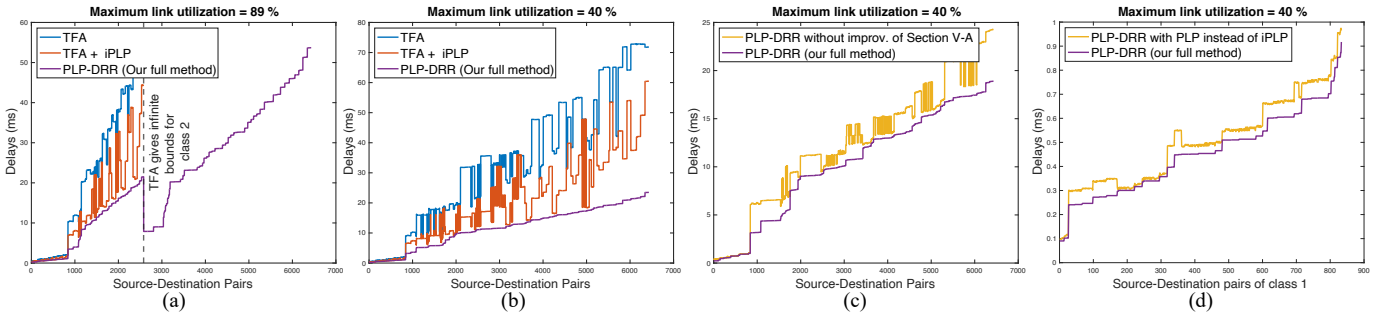


Fig. 6: Delay bounds of our method compared to alternative methods. (a-b): comparison with 1)-2). At link utilization 89%, TFA analysis gives infinite delays for class 2. (c-d): effect of the two PLP improvements of Section V. Source-destination paths are ordered by values of our full method, except in (a) where finite delays for TFA are shown first.

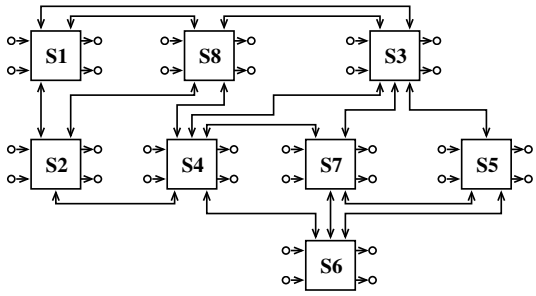


Fig. 7: Industrial-sized network topology. Figure from [23].

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. 6 (d), 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.

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 VI-B. We provide the 95% confidence interval for the run-times of 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%: The initial phase takes $[0.5,\ 0.6]$ minutes; the versions 1 and 2 of refinements phase takes $[5.8,\ 5.9]$ and $[4.6,\ 4.7]$ minutes, respectively. When the maximum link utilization is 89%: The initial phase takes $[3,\ 4]$ minutes; the versions 1 and 2 of refinements phase takes $[9,\ 10]$ and $[5.5,\ 6.5]$ minutes, respectively. Regarding 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: For PLP we obtain $[5.7,\ 5.75]$ seconds and for iPLP we obtain $[6.4,\ 6.5]$ seconds.

IX. 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 on 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. It will be interesting to study other concrete implementations that aim at reducing the computing time.

References

[1] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, pp. 375–385, 1996.

[2] 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)*, 2002, pp. 77–86.

[3] S. M. Tabatabaee and J.-Y. L. Boudec, "Deficit round-robin: A second network calculus analysis," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021, pp. 171–183.

[4] S. S. Kanhere and H. Sethu, "On the latency bound of deficit round robin," in *Proceedings. Eleventh International Conference on Computer Communications and Networks*, 2002, pp. 548–553.

[5] D. Stiliadis, "Traffic scheduling in packet-switched networks: Analysis, design, and implementation," Ph.D. dissertation, 1996, aAI9637506.

[6] L. Lenzini, E. Mingozzi, and G. Stea, "Full exploitation of the deficit round robin capabilities by efficient implementation and parameter tuning."

[7] 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*, 2012, pp. 138–147.

[8] A. Bouillard, "Individual service curves for bandwidth-sharing policies using network calculus," *IEEE Networking Letters*, vol. 3, no. 2, pp. 80–83, 2021.

[9] A. Bouillard, M. Boyer, and E. Le Corronc, *Deterministic Network Calculus: From Theory to Practical Implementation*. Wiley-ISTE.

[10] L. Thomas, J.-Y. Le Boudec, and A. Mifdaoui, "On cyclic dependencies and regulators in time-sensitive networks," in *2019 IEEE Real-Time Systems Symposium (RTSS)*, 2019, pp. 299–311.

[11] S. Plassart and J.-Y. Le Boudec, "Equivalent versions of total flow analysis," *CoRR*, vol. abs/2111.01827, 2021. [Online]. Available: https://arxiv.org/abs/2111.01827

[12] A. Bouillard, "Trade-off between accuracy and tractability of network calculus in FIFO networks," *Perform. Eval.*, vol. 153, no. C, feb 2022. [Online]. Available: https://doi.org/10.1016/j.peva.2021.102250

[13] A. Scheffler and S. Bondorf, "Network calculus for bounding delays in feedforward networks of fifo queueing systems," in *Quantitative Evaluation of Systems*, A. Abate and A. Marin, Eds. Cham: Springer International Publishing, 2021, pp. 149–167.

[14] F. Geyer, A. Scheffler, and S. 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)*, 2021, pp. 157–170.
[15] 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*, Aug 2008, pp. 387–392.
[16] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *2000 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 4, May 2000, pp. 101–104 vol.4.
[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.
[18] S. M. Tabatabaee and J.-Y. Le Boudec, "Deficit round-robin: A second network calculus analysis," *IEEE/ACM Transactions on Networking*, pp. 1–15, 2022.
[19] J. B. Schmitt and F. A. Zdarsky, "The disco network calculator: A toolbox for worst case analysis," in *Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools*, ser. valuetools '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 8–es. [Online]. Available: https://doi.org/10.1145/1190095.1190105
[20] A. Mifadoui and T. 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)*, Paris, France, Dec. 2017, pp. pp. 1–8. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01690096
[21] J. Grieu, "Analyse et évaluation de techniques de commutation ethernet pour l'interconnexion des systèmes avioniques," September 2004. [Online]. Available: https://oatao.univ-toulouse.fr/7385/
[22] A. Bouillard, "Stability and performance bounds in cyclic networks using network calculus," in *Formal Modeling and Analysis of Timed Systems - 17th International Conference, FORMATS 2019, Amsterdam, The Netherlands, August 27-29, 2019, Proceedings*, ser. Lecture Notes in Computer Science, É. André and M. Stoelinga, Eds., vol. 11750. Springer, 2019, pp. 96–113. [Online]. Available: https://doi.org/10.1007/978-3-030-29662-9_6
[23] 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)*, 2006, pp. 10 pp.–202.
[24] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer Science & Business Media, 2001, vol. 2050.
[25] C. S. Chang, *Performance Guarantees in Communication Networks*. New York: Springer-Verlag, 2000.

APPENDIX A
DETAILED BACKGROUND ON DRR STRICT SERVICE CURVES [3], [18]

Here we present more background on DRR strict service curves of [18], using our notations, that enables a reader to implement what we use in the paper.

A. Degraded Operational Mode

Here we present Corollary 2 of [18] 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 II-A, 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, $d_c^{\max,v}$ is the maximum residual deficit defined by $d_c^{\max,v} = l_c^{\max,v} - \epsilon$ where $l_c^{\max,v}$ 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 $\beta_c^{\text{CDM},v}$ given by $\beta_c^{\text{CDM},v}(t) = \gamma_c^{\text{convex},v}\left(B^v(t)\right)$ with

$$\gamma_c^{\text{convex},v} = \max\left(\beta_{R_c^{\max,v},T_c^{\max,v}}, \beta_{R_c^{\min,v},T_c^{\min,v}}\right) \tag{6}$$

$$R_c^{\max,v} = \frac{Q_c^v}{Q_{\text{tot}}^v},\ T_c^{\max,v} = \sum_{c',c'\neq c}\left(Q_{c'}^v + d_{c'}^{\max,v} + \frac{Q_{c'}^v}{Q_c^v} d_c^{\max,v}\right) \tag{7}$$

$$R_c^{\min,v} = \frac{Q_c^v - d_c^{\max,v}}{Q_{\text{tot}}^v - d_c^{\max,v}},\ T_c^{\min,v} = \sum_{c',c'\neq c}\left(Q_{c'}^v + d_{c'}^{\max,v}\right) \tag{8}$$

$$Q_{\text{tot}}^v = \sum_c Q_c^v \tag{9}$$

In (6), $\beta_{R,T}$ is a rate-latency function defined in Table I.

For the non-convex strict service curve, $\beta_c^{\text{nc},v}$, we have

$$q_c^v = Q_c^v - d_c^{\max,v} \tag{10}$$

$$T_1 = T_c^{\min,v} \tag{11}$$

B. Non-Degraded Operational Mode

Here we present a new formulation of Corollary 4 of [18]. We slightly generalize Corollary 4 of [18], using Lemma 2 of [18], such that it enables us to take into account any available output arrival curves. Specifically, Corollary 4 of [18] is an application of this new formulation where we replace $\alpha_{c'}^*$ with $\alpha_{c'} \oslash \beta_{c'}^{\text{old},v}$, which is an output arrival curve for class c', in (12). Note that $\oslash$ is the min-plus deconvolution defined by $(y \oslash y')(t) = \sup_{s\geq 0}\{y(t+s) - y'(s)\}$ for two non-decreasing functions y and y' [9], [24], [25].

Let v be a node with the assumptions in Section A-A. Also, assume that every class c has an output arrival curve α_c^* and a strict service curve $\beta_c^{\text{old},v}$, 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 $\beta_c^{\text{new},v}$ is given by

$$\beta_c^{\text{new},v} = \max\left(\beta_c^{\text{old},v}, \max_{J\subseteq N_c} \gamma_c^{\text{convex},v\,J} \circ \left[B^v - \sum_{c'\in\bar{J}} \alpha_{c'}^*\right]_\uparrow^+\right) \tag{12}$$

with

$$\gamma_c^{\text{convex},v\,J} = \max\left(\beta_{R_c^{\max,v\,J},T_c^{\max,v\,J}}, \beta_{R_c^{\min,v\,J},T_c^{\min,v\,J}}\right) \tag{13}$$

$$R_c^{\max,v\,J} = \frac{Q_c^v}{Q_{\text{tot}}^{J,c}},\ T_c^{\max,v\,J} = \sum_{c'\in J}\left(Q_{c'}^v + d_{c'}^{\max,v} + \frac{Q_{c'}^v}{Q_c^v} d_c^{\max,v}\right) \tag{14}$$

$$R_c^{\min,v\,J} = \frac{Q_c^v - d_c^{\max,v}}{Q_{\text{tot}}^{J,c} - d_c^{\max,v}},\ T_c^{\min,v\,J} = \sum_{c'\in J}\left(Q_{c'}^v + d_{c'}^{\max,v}\right) \tag{15}$$

$$Q_{\text{tot}}^{J,c} = Q_c + \sum_{c'\in J} Q_{c'} \tag{16}$$

In (12), $[.]_\uparrow^+$ 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 (13), $\beta_{R,T}$ is a rate-latency function defined in Table I.

The essence of (12) is as follows. Equation (12) gives new strict service curves $\beta_c^{\text{new},v}$ for every flow c; they are derived from already available strict service curves $\beta_c^{\text{old},v}$ 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 $\Pi_v^{\text{convex}} : \mathscr{F}^{2n} \to \mathscr{F}^n$ be the mapping at server v that maps $\left(\beta_1^{\text{old},v}, \beta_2^{\text{old},v}, \ldots, \beta_n^{\text{old},v}\right)$ using $(\alpha_1^*, \alpha_2^*, \ldots, \alpha_n^*)$ to $(\beta_1^{\text{new},v}, \beta_2^{\text{new},v}, \ldots, \beta_n^{\text{new},v})$ as in (12). Then, an iterative scheme can be defined as in Algorithm 3. Note that Algorithm 3 is exactly the same as Algorithm 2 of [18], adopted with the new formulation that is presented in (12).

Algorithm 3: $\text{DRRservice}_v^{\text{inputArrival}}(\alpha_1, \ldots, \alpha_n)$

Result: Collection of strict service curves $(\beta_1^v, \ldots, \beta_n^v)$
Local Variables: Collections of strict service curves $\left(\beta_1^{\text{old},v}, ..., \beta_n^{\text{old},v}\right)$ and $(\beta_1^{\text{new},v}, ..., \beta_n^{\text{new},v})$

1 **for** $c \leftarrow 1$ *to* n **do**
2 $\quad \beta_c^{\text{old},v} \leftarrow \beta_c^{\text{CDM},v}$;
3 **while** *Stopping criteria not reached* **do**
4 $\quad$ **for** $c \leftarrow 1$ *to* n **do**
5 $\quad\quad \alpha_c^* \leftarrow \alpha_c \oslash \beta_c^{\text{old},v}$;
6 $\quad \beta^{\text{new},v} \leftarrow \Pi_v^{\text{convex}}\left(\alpha^*, \beta^{\text{old},v}\right)$;
7 $\beta^v \leftarrow \beta^{\text{new},v}$;
8 **return** $(\beta_1^v, \ldots, \beta_n^v)$

It is shown in [18] that for every class c , $\beta_c^{\text{old},v}$ and $\beta_c^{\text{new},v}$ are strict service curves for class c and $\beta_c^{\text{old},v} \leq \beta_c^{\text{new},v}$, 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 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 $\beta_c^{\text{CDM},v}$, obtained in Section A-A for the initial strict service curves at lines 1-2.

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, $\text{DRRservice}_v^{\text{outputBurst}}(\beta^v, b^v)$ is the function that implements (12) (i.e., Π_v^{convex} defined above) and returns an improved collection of strict service curves for all classes at node v.

APPENDIX B
DETAILED BACKGROUND ON PLP [12]

Here we present more background on PLP of [12], using our notations, that enables a reader to implement what we use in the paper. Specifically, we summarize linear programs used by PLP. For the rest of the section, a reader is invited to recall the definitions of Section V.

Note that linear programs of [12] contain some constraints obtained from the delay bound obtained using Single Flow Analysis (SFA) [9]. However, in practice, such constraints have no or negligible effects as SFA bounds are often dominated by those of TFA. Hence, in this paper, we do not use SFA constraints.

A. $PLP_{f,c}^{delay}$: A PLP That Computes an End-to-end Delay Bound for a Single Flow

Here we summarize linear programs used by PLP, and more details can be found in [12, Section 4].

The goal of $\text{PLP}_{f,c}^{\text{delay}}(\beta_c, d_c, z_c^{\text{cut}})$ 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 are 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.

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:
- $\forall v \in \mathcal{V}_c^f,\ \forall k \in [0,\ \text{dp}(v) - 1], \mathbf{t}_{(v,k)} \geq \mathbf{t}_{(v,k+1)}$;
- $\forall v \in \mathcal{V}_c^f,\ \forall k \in [0,\ \text{dp}(v)], \mathbf{t}_{(v,k)} \leq \mathbf{t}_{(u,k)}$.

• FIFO Constraints:
- $\forall v \in \mathcal{V}_c^f,\ \forall k \in [0,\ \text{dp}(u)], \forall g \in \text{In}_c(v), \mathbf{Ft}_{v,k}^{v,g} = \mathbf{Ft}_{u,k}^{u,g}$.

• 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 $\mathbf{At}_u^v = \sum_{g \in \text{In}(v)} \mathbf{Ft}_{u,\text{dp}(u)}^{u,g}$ and $\mathbf{At}_v^v = \sum_{g \in \text{In}(v)} \mathbf{Ft}_{v,\text{dp}(v)}^{v,g}$, where $u = \text{sc}(v)$. Then, $\forall v \in \mathcal{V}_c^f$,
- $\mathbf{At}_u^v - \mathbf{At}_v^v \geq 0$;
- $\forall p,\ \mathbf{At}_u^v - \mathbf{At}_v^v \geq R_p^v \left(\mathbf{t}_{(u,\text{dp}(u))} - \mathbf{t}_{(v,\text{dp}(v))} - T_p^v\right)$.

• Per-node Delay Bound Constraints:
- $\forall v \in \mathcal{V}_c^f,\ \forall k \in [0,\ \text{dp}(u)],\ \mathbf{t}_{(u,k)} - \mathbf{t}_{(v,k)} \leq d_c^v$.

• 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 II, 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,
- $\forall 0 \leq k < k' \leq \text{dp}(v_g),\ \mathbf{Ft}_{v_g,k}^{v_g,g} - \mathbf{Ft}_{v_g,k'}^{v_g,g} \leq \bar{b}_g + r_g \left(\mathbf{t}_{(v_g,k)} - \mathbf{t}_{(v_g,k')}\right)$.

• 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,
- $\forall 0 \leq k < k' \leq \text{dp}(u),\ \sum_{g \in F_{v,u}} \left(\mathbf{Ft}_{u,k}^{u,g} - \mathbf{Ft}_{u,k'}^{u,g}\right) \leq l_c^{\max,v} + R^v \left(\mathbf{t}_{(u,k)} - \mathbf{t}_{(u,k')}\right)$.

• 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,
- $\forall k \in [0, \text{dp}(v_g) - 1],\ \mathbf{Ft}_{v_g,k}^{v_g,g} \geq \mathbf{Ft}_{v_g,k+1}^{v_g,g}$.

2) Objective: The Objective is $\max\left(\mathbf{t}_{(v_0,0)} - \mathbf{t}_{(v_f,0)}\right)$.

B. $PLP^{backlog}_{f,c}$: 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 [12, Theorem 4], the objective of this linear program, is a bound on the burstiness of flow f at the output of node v.

1) Constraints: This linear program contains all the constraints of $\text{PLP}^{\text{delay}}_{f,c}$, defined in Section B-A, 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.

- $\forall k \in [0, \text{dp}(v_f)]$, $\mathbf{Ft}^{v_f,f}_{v_0,0} - \mathbf{Ft}^{v_f,f}_{v_f,k} \leq b_f + r_f\left(\mathbf{t}_{(v_0,0)} - \mathbf{t}_{(v_f,k)}\right)$.

2) Objective: The Objective is $\max\left(\mathbf{Ft}^{v_f,f}_{v_0,0} - \mathbf{Ft}^{v_0,f}_{v_0,0}\right)$.

C. FP-PLP_c: A PLP That Computes Bounds on The Burstiness of Flows at Cuts

The goal of $\text{FP-PLP}_c(\beta_c, d_c)$ 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, piecewise 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 B-B, 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 [12, 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.

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 B-B. 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 $\mathbf{x}_g$.

2) Objective: The Objective is $\max \sum_{f \in F_c^{\text{cut}}} \mathbf{x}_f$.