Beyond the Accuracy-Complexity Tradeoffs of Compositional Analyses using Network Calculus for Complex Networks

Ahlem Mifdaoui, Thierry Leydier

▶ **To cite this version:**

Ahlem Mifdaoui, Thierry Leydier. Beyond the Accuracy-Complexity Tradeoffs of Compositional Analyses using Network Calculus for Complex Networks. 10th International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (co-located with RTSS 2017), Dec 2017, Paris, France. pp. 1-8. hal-01690096

HAL Id: hal-01690096
https://hal.science/hal-01690096v1

Submitted on 22 Jan 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Open Archive Toulouse Archive Ouverte (OATAO)

OATAO is an open access repository that collects the work of some Toulouse researchers and makes it freely available over the web where possible.

This is an author's version published in: https://oatao.univ-toulouse.fr/19256

Official URL:

To cite this version :

Mifdaoui, Ahlem and Leydier, Thierry Beyond the Accuracy-Complexity Tradeoffs of Compositional Analyses using Network Calculus for Complex Networks. (2017) In: 10th International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (co-located with RTSS 2017), 5 December 2017 - 5 December 2017 (Paris, France).

Any correspondence concerning this service should be sent to the repository administrator:
tech-oatao@listes-diff.inp-toulouse.fr

Beyond the Accuracy-Complexity Tradeoffs of Compositional Analyses using Network Calculus for Complex Networks

Ahlem MIFDAOUI
University of Toulouse/ ISAE-SUPAERO
ahlem.mifdaoui@isae.fr

Thierry LEYDIER
Virtualité Réelle
thierry.leydier@virtualitereelle.com

ABSTRACT

Achieving the accuracy-complexity tradeoffs for compositional timing analyses using Network Calculus is still a hot research topic. In this specific area, we propose in this paper an improved version of the Total Flow Analysis (TFA) algorithm, called TFA++, when taking into account the impact of the finite transmission capacity of the network links on the input and output traffic models at each network node. First, we review the existing analysis algorithms by identifying their main limitations in terms of accuracy and complexity, through a simple but representative network example. Afterwards, we define the TFA++ algorithm and we detail the main steps of the followed methodology to compute the delay upper bounds. Moreover, we conduct comparative analyses of the derived delay bounds and analysis times with the different algorithms, with respect to the network size and load. In doing this, we highlight noticeable enhancements of both metrics under TFA++, in comparison to the existing algorithms; thus the high accuracy and low complexity of TFA++. Finally, this statement has been asserted through a representative avionics case.

KEYWORDS

Network Calculus, Pay Multiplex Only Once (PMOO), Performance analysis, Delay bounds, Accuracy-complexity trade-off, Total Flow Analysis (TFA)

ACM Reference format:
Ahlem MIFDAOUI and Thierry LEYDIER. 2017. Beyond the Accuracy-Complexity Tradeoffs of Compositional Analyses using Network Calculus for Complex Networks. In *Proceedings of The 10th International Workshop on Compositional Theory, Paris, France, 2017 (CRTS),* 8 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

With the inherent complexity and the growing number of components, in addition to the use of advanced communication technologies and multi/many core processors, many challenges have emerged from designing new generation Cyber-Physical Systems (CPS). Hence, innovative cost-effective methods and tools are needed to design, analyze and verify the hardware and software architecture components of such systems, to guarantee predictability and reliability requirements while minimizing the development and maintenance costs. Particularly, for new generation CPS, the communication networks are considered as a bottleneck for performance and timing predictability; thus an appropriate performance analysis to provide safe guarantees has to be considered.

Most performance analysis approaches of complex networks can broadly be categorized under two main classes, simulation-based and analytical-based. The former enables the performance analysis of large-scale networks, but it is not appropriate to draw firm conclusions since it does not cover all corner cases, i.e., worst-case scenario; whereas, the latter is based on high-level analytical models that can be easily injected within optimization procedures, to speed up the design space exploration. These facts make the use of analytical-based approaches outwardly growing for early verification of networks in CPS, and one of the most relevant approaches in this specific area is the *Network Calculus* [9]. The *Network Calculus* is a compositional algebraic framework to derive maximum bounds on system performance, such as delays and backlogs. The high modularity and scalability of such a framework make it particularly efficient to conduct performance analysis of complex communication networks [12], such as Switched Ethernet [11], the AFDX[8], Networks on Chip [13] and networks with cyclic dependencies [2].

However, one of the main challenging issues for Network Calculus is computing accurate performance bounds with a reasonable time-effort. Particularly, the lack of accuracy may lead to resource over-dimensioning of the networks under design; thus increasing development costs. Most of the related work in this area is focusing on improving the timing analysis algorithms [10] [15] [4], to cope with the accuracy-complexity tradeoffs. Such research efforts lead to two main classes of approaches using Network Calculus: algebraic and optimization-based methods. In this paper, our main focus is the algebraic class, since it is the one keeping the composition property of Network Calculus. Particularly, there are mainly three existing timing analysis algorithms [14]: Total Flow Analysis (TFA), Separated Flow analysis (SFA) and Pay Multiplex Only Once (PMOO). Each one of these aforementioned algorithms has its pros and cons in terms of accuracy and complexity, but there is no best algorithm in terms of both metrics.

To overcome these limitations, we propose in this paper an improved TFA algorithm (TFA++) to enhance accuracy, while keeping low complexity. The main idea of such an improved algorithm consists in considering the link transmission capacity impact on the input and output traffic models of each crossed node in the network.

Hence, the main contributions in this paper are as follows:
(i) First, we review the existing analysis algorithms by identifying their main limitations in terms of accuracy and complexity, through a simple but representative network example. In doing this, we consolidate the idea of the non-existence of a best algorithm;

(ii) Second, we define the TFA++ algorithm (Algorithm 1) and we detail the main steps of the followed methodology to compute the delay upper bounds;
(iii) Third, we conduct comparative analyses of the derived delay bounds and analysis times with the different algorithms, with respect to the network size and load. In doing this, we highlight noticeable enhancements of both metrics under TFA++, in comparison to the existing algorithms; thus the high accuracy and low complexity of TFA++. Finally, this statement has been asserted through a representative avionics case.

The rest of the paper is organized as follows: we start with presenting the main concepts of the Network Calculus framework in Section 2. Afterwards, we report the main related work and the identified limitations in Section 3. Then, we present the proposed TFA++ algorithm and report evaluation results in Sections 4 and 5, respectively. Finally, we draw the main conclusions and future work in Section 6.

2 THE NETWORK CALCULUS AS A COMPOSITIONAL ALGEBRAIC THEORY

The Network Calculus framework has been founded by the seminal work of Cruz in [6, 7], and then extended with min-plus Algebra operations in [5] and [9]. The latter extension is based on the idea of modeling the communication nodes as in conventional system theory, with an input function, a transfer function and an output function, where addition and multiplication are replaced by minimum and addition, respectively. Particularly, we will detail in this section how this algebraic extension has enabled the composition property of Network Calculus.

We will answer herein some primordial questions when applying Network Calculus to conduct performance analysis of a realistic network: how to model the input traffic? how to model the node specifications? how to deal with a network of nodes to compute end-to-end performance?

2.1 Traffic Model

Network Calculus describes data flows by means of cumulative functions, $R(t)$, defined as the number of transmitted bits during the time interval $[0, t]$. Function $R(t)$ is always a non-decreasing function of time with $R(0) = 0$. Consider a system S receiving input data with cumulative function $R(t)$, called input function. After the data processing, the output data is described using another cumulative function $R^*(t)$, called output function. The horizontal distance $d(t)$ between the input and output functions represents the experienced delay of an input data received at time t in the system. The vertical distance $x(t)$ between input and output functions represents the backlog, i.e., total number of bits present in the system at instant t.

One of the fundamental concepts in Network Calculus is the notion of maximum arrival curve, which provides an upper bound on the number of events, e.g., bits or packets, observed during any interval of time, as shown in Figure 1. Consequently, this curve covers all possible scenarios of traffic arrivals, observed at any instant, i.e., if α is a maximum arrival curve, then for any interval duration Δ, there will be at most $\alpha(\Delta)$ events. This concept allows modeling a large panel of event arrival patterns, such as periodic, sporadic, with or without jitter, and bursty or not. The definition of the arrival curve is as following:

Definition 2.1. (Arrival Curve)[9] A function α is an arrival curve for a data flow with an input cumulative function R, such that $R(t)$ is the number of bits received until time t, iff:

$$\forall t, s \geq 0, s \leq t, R(t) - R(s) \leq \alpha(t - s) \quad (1)$$

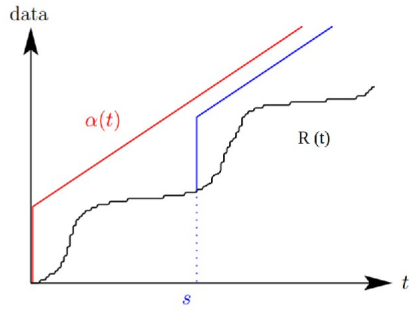

Figure 1: *Arrival curve*

The considered arrival patterns necessary to define the arrival curve can be obtained from traffic traces if any, or formal specification. The latter is more common for real-time communication systems under design. The network designer generally specifies a traffic contract for each application, that is enforced using a controller, e.g., policers or shapers. The most common controller follows a leaky bucket algorithm, which guarantees for the controlled traffic a maximum burst b and a maximum rate r on the communication medium, i.e., the traffic flow is (b, r)-constrained. In this case, the arrival curve is an affine curve, defined as $\lambda_{b,r}(t) = b + r.t$ for $t > 0$. Furthermore, there is the following interesting result concerning the arrival curves.

Lemma 2.2. *[9] If α_1 and α_2 are arrival curves for a flow R, then so is $\alpha_1 \otimes \alpha_2$, where $(f \otimes g)(t) = \inf_{0 \leq s \leq t}\{f(t - s) + g(s)\}$.*

2.2 Node Model

To conduct worst-case performance analysis, we need to put constraints on the input traffic through the maximum arrival curve notion. In return, we need to guarantee a minimum offered service within crossed nodes to cover the worst-case behavior and infer upper bounds on performance metrics, e.g., backlog and delay. This is done through the concept of minimum service curve, which is defined as following.

Definition 2.3. [9] (Simple Minimum Service Curve) The function β is the simple service curve for a data flow with an input cumulative function R and output cumulative function R^* iff:

$$\forall t \geq 0, R^*(t) \geq \inf_{s \leq t}(R(t) + \beta(t - s)) \quad (2)$$

A very useful and common model of service curve is the rate-latency curve $\beta_{R,T}$, with R the minimum guaranteed rate and T the maximum latency before starting the service. This rate-latency function is defined as follows: $\beta_{R,T}(t) = 0$ if $t \leq T$ and $R(t - T)$ otherwise.

This service curve is easy to define in the case of *one input/ output node* serving one or many traffic flows coming from the same source and going to the same destination. However, to handle more realistic scenarios with a network of nodes, implementing aggregate scheduling which multiplexes the crossing flows at the input and demultiplexes them at the output, we need to define the *left-over service curve* guaranteed to each traffic flow within each crossed node, considering the impact of the other traffic flows in contention, to infer the offered guarantees for each flow. The computation of such a left-over service curve depends on the implemented scheduling policy within each crossed node, and the most common ones are Blind Multiplexing, FIFO and Fixed Priority (FP). It is worth noting that this derivation needs strict service curve property in the general case, except for FIFO and Constant bit rate nodes. A minimum strict service curve is defined as follows.

Definition 2.4. [9] (Strict service curve) The function β is the strict service curve for a data flow with an input cumulative function R and output cumulative function R^*, if for any backlogged period[1] $]s,t]$, $R^*(t) - R^*(s) \geq \beta(t-s)$.

The main result concerning the left over service curves computation is as follows:

Theorem 2.5 (Residual service curve - Blind Multiplex). *[3] let f_1 and f_2 be two flows crossing a server that offers a strict service curve β such that f_1 is α_1-constrained, then the residual service curve offered to f_2 is:*

$$\beta_2 = (\beta - \alpha_1)_\uparrow$$

where $f_\uparrow(t) = \max\{0, \sup_{0\leq s\leq t} f(s)\}$

2.3 Performance Analysis

Knowing the arrival and service curves, one may compute the performance bounds for data flows, e.g., delay and backlog. In the case of single node with *one input/ output*, these bounds are computed according to the following theorem.

Theorem 2.6 (Performance Bounds). *[9] Consider a flow constrained by an arrival curve α crossing a system $\mathcal{S}$ that offers a minimum service curve β and a maximum service[2] curve γ. The performance bounds obtained at any time t are given by:*
Output arrival curve[3] *:* $\alpha^*(t) = \alpha \oslash \beta(t)$
Tight Output arrival curve: $\alpha^*(t) = (\gamma \otimes \alpha) \oslash \beta(t)$
Backlog[4]*:* $\forall\ t:\ q(t) \leq v(\alpha, \beta)$
Delay[5]*:* $\forall\ t:\ d(t) \leq h(\alpha, \beta)$

The calculus of these bounds is greatly simplified in the case of leaky bucket arrival curve and the Rate-Latency service curve. In this case, the delay is bounded by $\frac{b}{R} + T$, the backlog bound is $b + r * T$, and the output arrival curve is $b + r(T + t)$. These results are illustrated in Figure 2.

To extend this result to a *network of nodes*, one of the strongest result in the Network Calculus framework is the computation of an end-to-end service curve for a tandem of nodes crossed by the same flows. This curve is computed as the convolution of residual service curves in each node, computed according to Theorem 2.6. This result is described in the following theorem.

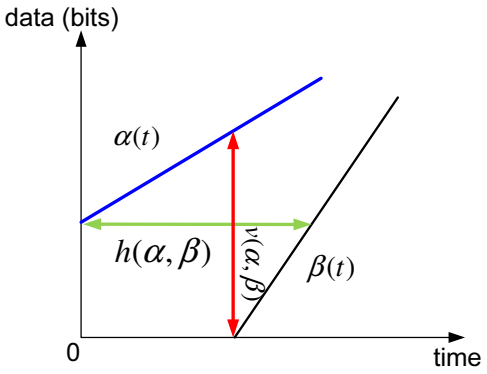

Figure 2: Backlog and Delay Bounds

Theorem 2.7 (Concatenation-Pay Bursts Only Once). *[9] Assume a flow crossing two servers with respective service curves β_1 and β_2. The system composed of the concatenation of the two servers offers a minimal service curve $\beta_1 \otimes \beta_2$ to the flow.*

Furthermore, this result infer an interesting property known as "Pay bursts Only Once Phenomena". Indeed, the end-to-end delay bound for a data flow, computed using the end-to-end service curve obtained with Theorem 2.7, clearly outperforms the sum of delay bound per node, computed iteratively using Theorem 2.6 and denoted as additive delay bound. The computation of these two bounds show the appearance of the burst term many times in the additive delay bound, as opposed to only once for the other.

3 CONVENTIONAL TIMING ANALYSIS ALGORITHMS AND LIMITATIONS

In the research community, there has been a growing interest in the subject of accuracy-complexity tradeoffs of performance analysis approaches using Network Calculus and we particularly focus herein on the compositional algebraic approaches. There are three main algorithms to compute the end-to-end delay bound for each flow of interest (f.o.i). All of them need to start the procedure by defining the initial input arrival curve within the network, i.e., within the source nodes, and the service curve guaranteed within each crossed node to the aggregate flows. Then, the main difference between the different algorithms relies on how to use these traffic and node models to derive the end-to-end delay bounds.

Total Flow Analysis (TFA) [14]
This algorithm consists in computing iteratively the delay upper bound in each crossed node for the aggregate flow, then the sum results in end-to-end delay bounds [7]. The main steps are:
(i) summing up all the arrival curves of individual flows at the input of each crossed node;
(ii) computing the delay bound based on Theorem 2.6, when considering the arrival curve of the aggregate flows and the global service curve guaranteed within the node;

[1] A backlogged period $]s,t]$ is an interval of time during which the backlog is non null, i.e., $R(s) = R^*(s)$ and $\forall u \in]s,t]$, $R(u) - R^*(u) > 0$
[2] $\forall t,\ R^*(t) \leq R \otimes \gamma(t)$
[3] $f \oslash g(t) = \sup_{s\geq 0}\{f(t+s) - g(s)\}$
[4] $v(f,g)$ is the maximum vertical distance between f and g
[5] $h(f,g)$ is the maximum horizontal distance between f and g

(iii) computing the output arrival curve of each flow using Theorem 2.6, when considering the left-over service curve based on Theorem 2.5.

Separated Flow Analysis (SFA) [14]

This algorithm consists in considering the end-to-end service curve, through the concatenation of the left over service curves, guaranteed for the f.o.i within each crossed node (applying Theorem2.7). This method is also based on three steps:

(i) summing up the input arrival curves of flows in contention with the f.o.i;

(ii) computing the left over service curve guaranteed to the f.o.i, based on Theorem 2.5;

(iii) computing the output arrival curve of the f.o.i.

These steps need to be conducted for each flow within each crossed node, to infer all the input arrival curves for the next node. This procedure has to be continued along each flow path to have all the left over service curves; thus the end-to-end service curve.

Pay Multiplex Only Once (PMOO) [16]

The main idea of this algorithm is based on accounting the flow serialization phenomena along the flow path to compute tighter end-to-end delay bounds. This fact consists in defining a smart application order of Theorem 2.7 and Theorem 2.5 to start concatenating as much as possible the service curves of nodes, crossed by the same group of interfering flows with the f.o.i (applying Theorem2.7); and then to compute the left-over service curve of the concatenated system guaranteed to the f.o.i (applying Theorem 2.5).

In doing this, we pay only once the bursts of interfering flows with the f.o.i along its path.

Limitations and Discussion

To better evaluate these algorithms, we conduct a brief comparative analysis of derived delay bounds and analysis times under the different algorithms, when increasing the network load for a simple but representative network example, shown in Figure 3.

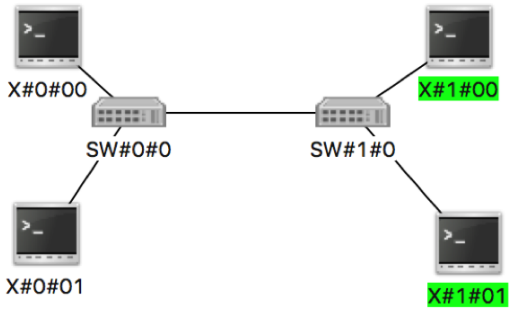

Figure 3: Illustrative Network Example

As shown in Figure 3, the network example consists of two identical 100Mbps full-duplex ethernet switches, implementing the store and forward technique and FIFO policy with a negligible technological latency; and four stations where each one generates one periodic flow with a period $P = 128$ms and a maximum packet length $L = 267$ bytes, sent in broadcast through the network.

First, each ethernet switch is modeled with a rate-latency service curve, $\beta_{C,T}$ with $C = 10^8$Mbps and $T = L/C = 21 * 10^{-6}$s.

Afterwards, each generated flow is modeled with a leaky-bucket arrival curve, $\lambda_{b,r}$ with $b = L = 2136$bits and $r = L/P = 16,68 * 10^3$bit/s.

To conduct the comparative analysis of the accuracy and complexity of the different algorithms, we vary the network load U through increasing the number of generated flows per station n; thus the network load percentage $U = n * r$ is varying in $[3 : 3 : 50]$.

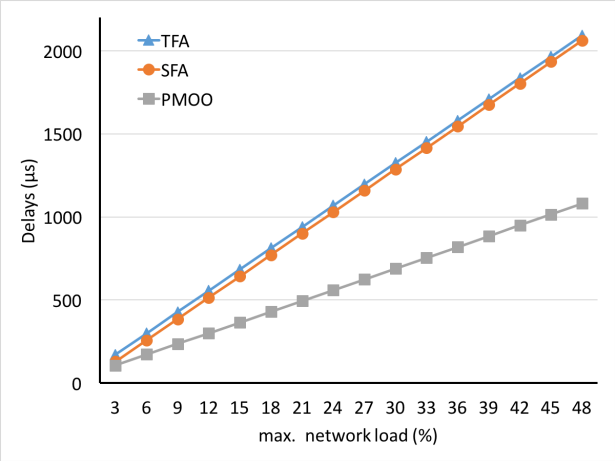

Figure 4: Delay bounds vs Network load of the different algorithms for the Network example

The derived delay bounds and analysis times under the different algorithms regarding the network load are illustrated in Figure 4 and Table 1, respectively. As we can notice, the delay bounds increase linearly with the network load. Furthermore, the PMOO guarantees the lowest delay bounds, which confirms the high accuracy of PMOO in comparison with TFA and SFA algorithms. On the other hand, the TFA offers the lowest maximum analysis time per flow, which shows the low complexity of TFA in comparison with SFA and PMOO. These results consolidate our statement that there is no best algorithm in terms of both metrics.

	TFA	SFA	PMOO
Max. Analysis Time per flow (us)	897	1683	953

Table 1: Maximum Analysis Times per flow of the different algorithms for the Network example

Therefore, there is a need of a new solution to bridge the gap between these existing algorithms and guarantee high accuracy and low complexity. To achieve this aim, we make the choice of improving the TFA algorithm to favor the time-effort metric, and this extension is detailed in the next section.

4 IMPROVED TFA ALGORITHM

In this section, we present the main idea of our proposed algorithm, called TFA++, to compute accurate delay bounds with low complexity. Afterwards, we detail the main steps of the computation methodology and explain the algorithm. Finally, we express the delay bounds for an illustrative network example under TFA++ and PMOO, to show the expected enhancement in terms of accuracy, while keeping a low complexity.

4.1 Main Idea

Without loss of generality, we model the network as a direct graph where the nodes are the output ports of the different network components and the edges the network links with finite transmission capacities. It is worth noting that the output ports are generally the only network parts inducing unknown multiplexing delays; whereas the other kinds of ports or blocks generally infer maximum constant delays that we can easily add at the end of the delay bound computation.

The main idea of the TFA++ is to take into account the impact of the link capacity on the traffic model to enhance the end-to-end delay bound tightness. Hence, we present herein two improvements due to this fact.

Tighter input arrival curve of an aggregate traffic
To refine the input arrival curve of an aggregate traffic at a given node with multiple inputs, where each input l has a finite transmission capacity C_l, we define the following notations:

- $i \ni k$ the set of flows crossing the node k;
- Each flow i has an input arrival curve at each crossed node k, $\alpha_i^{k\ominus 1}(t)$;
- $Pred_N$ is the set of nodes transmitting flows on the input of node N.

The aggregate traffic sent by each node $l \in Pred_N$ on the input of node N can be modeled through two possible arrival curves:

(1) the first one is simply the sum of individual flow arrival curves crossing l, $\sum_{i \ni l} \alpha_i^{N\ominus 1}$;
(2) knowing the finite transmission capacity of the link from node l to node N, C_l, we have a second arrival curve of this same aggregate traffic, λ_{0,C_l}.

Hence, based on Lemma 2.2, the aggregate traffic flow at the input of node N has the following arrival curve:

$$\sum_{l \in Pred_N} (\lambda_{0,C_l} \otimes \sum_{i \ni l} \alpha_i^{N\ominus 1}(t)) \tag{3}$$

According to Theorem 3.1.6 in [9], for functions passing through the origin, the convolution of both functions is at most equal to their minimum. Moreover, for the particular case of concave functions, e.g., leaky bucket curves, the convolution is equal to the minimum.

Hence, the refined input arrival curve in Eq. 3 used in TFA++ is necessarily tighter than the sum of the individual arrival curves, used in the original TFA algorithm:

$$\sum_{l \in Pred_N} (\lambda_{0,C_l} \otimes \sum_{i \ni l} \alpha_i^{N\ominus 1}(t)) \leq \sum_{l \in Pred_N} \sum_{i \ni l} \alpha_i^{N\ominus 1}(t) \tag{4}$$

Tighter output arrival curve of an individual flow
The finite transmission capacity at the output of a node N, C_N, infers a maximum service curve $\gamma^N(t) = C_N.t$.

Hence, through applying Theorem 2.6, we can compute the following output arrival curve from node N for each flow i, α_i^N, as follows:

$$\alpha_i^N(t) = (\alpha_i^{N\ominus 1} \otimes \gamma^N) \oslash \beta_i^N(t) \tag{5}$$

where $\beta_i^N(t)$ is the left-over service curve of the flow i at node N computed through applying Theorem 2.5.

The refined output arrival curve in Eq. (5) used in TFA++ is tighter than the one used in the original TFA algorithm:

$$(\alpha_i^{N\ominus 1} \otimes \gamma^N) \oslash \beta_i^N(t) \leq \alpha_i^{N\ominus 1} \oslash \beta_i^N(t) \tag{6}$$

In the next section, we explain the use of both improvements to compute the end-to-end delay bounds through the Algorithm 1.

4.2 Computation Methodology

Algorithm 1 Improved Total Flow Analysis ($Network$, F)

```
1:  for each flow of interest f ∈ F do
2:    for each sink ∈ sinks(f) do
3:      D_EED(f, sink) ← 0
4:      for each node N ∈ path_{src(f)=>sink}(f) do
5:        MaxDelay ← 0
6:        K ← set-of-flows-at-input (N, f)
7:        Pred ← Predecessors(N)
8:        MaxInput ← ComputeInput(N, f)
9:        MinService ← ComputeService(N, f)
10:       D ← ComputeDelay(MaxInput, MinService )
11:       for each flow k ∈ K do
12:         α_k^N(t) ← ComputeOutput(N, f)
13:       end for
14:       D_EED(f, sink) ← D_EED(f, sink) + D
15:     end for
16:     D_EED(f) ← Vector(D_EED(f, sink))
17:   end for
18:   D_EED ← Matrix(D_EED(f))
19: end for
20: return D_EED
```

The Delay bounds analysis based on the proposed TFA++ is described in Algorithm 1 for feed-forward networks, i.e., no cyclic dependencies, and supporting multicast communications.

For each flow of interest f (line 1) and for each one of its paths $path_{src(f)=>sink}(f)$ (line 2), the delay bound is computed within each crossed node N, accounting the set of flows in contention with f, K (line 6) and the the set of predecessors (line 7). Finally, the end-to-end delay is simply equal to the sum of these individual delays along the flow path (lines 4-15).

First, we compute the input arrival curve of the node N, considering the impact of the link capacity, as explained in Eq. (3) (line 8). Then, we compute the total service curve of the crossed node (line 9), to enable the computation of the delay bound within the crossed node through applying Theorem 2.6 (line 10). Afterwards, for each flow in the interference set, we need to compute the output arrival curve when taking into account the impact of the link capacity,

as explained in (Eq. 5) (lines 11-13). Finally, the end-to-end delay bound is the sum of all the crossed node delays until reaching the corresponding sink (line 16).

4.3 Illustrative Example

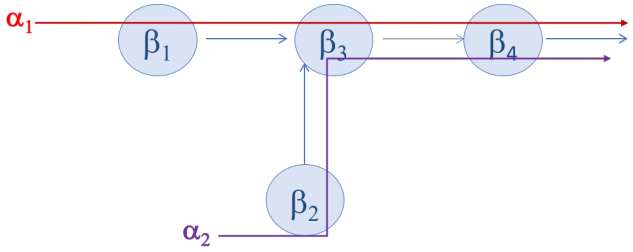

Figure 5: Illustrative Example

We consider herein the illustrative example in Figure 5 to express the derived delay bounds under TFA++ and PMOO for the flow of interest f_1 with the arrival curve α_1. We detail herein the delay bounds computation when considering identical flows, i.e., f_1 and f_2, with leaky-bucket arrival curves, $\lambda_{b,r}$, and nodes with rate-latency service curves, $\beta_{C,T}$.

Under PMOO, as we can notice, both flows are crossing the same nodes 3 and 4 until the final destination. Hence, we can start by concatenating the nodes 3 and 4 through applying Theorem 2.7, then computing the residual service curve offered to the f.o.i f_1 through Theorem 2.5. This combination gives the following end-to-end service curve under PMOO for f_1:

$$\beta_1 \otimes (\beta_3 \otimes \beta_4 - \alpha_2^{3\ominus 1})_{\uparrow} = \beta_{C-r,T+(2T.C+b+r.T)/C-r} \qquad (7)$$

Consequently, the end-to-end delay bound under PMOO when applying Theorem 2.6 is $D_{PMOO} = T + (2T.C + 2b + r.T)/(C - r)$.

Under TFA++, we compute the end-to-end delay bound as the sum of local delays in nodes 1, 3 and 4 crossed by the f.o.i. f_1. First, the delay bound in node 1 is simply equal to the maximum horizontal distance between the arrival and service curves; thus $D_1 = T + b/C$. Afterwards, to compute the delay in node 3, we define the input arrival curve of node 3 as in Eq. (3), $2*min(C.t, b+r(t+T))$, and consequently $D_3 = T + b/(C - r)$. Finally, we compute the output arrival curve of node 3 as in Eq. (5), $min(C.t, \alpha_1^{4\ominus 1} + \alpha_2^{4\ominus 1})$; thus $D_4 = T$. Hence, the end-to-end delay bound under TFA++ is $D_{TFA++} = 3T + b/(C - r) + b/C$.

As we can notice, $D_{TFA++} < D_{PMOO}$ for this illustrative example. This fact shows the promising accuracy of TFA++ in comparison to PMOO, and we will consolidate this statement in the next section under different network configurations.

5 EVALUATION

In this section, we first report the computed delay bounds and time analysis with respect to network size and load, to benchmark the existing timing analysis algorithms against our proposed TFA++ in terms of accuracy and complexity. Afterwards, we present the delay bounds for a representative avionics study to consolidate our evaluation. All the computed metrics in this section are based on the WoPANets tool [1].

5.1 Comparative Analyses

To conduct the comparative analyses, we consider the case study with the following assumptions:
(i) The topology is a mesh connecting M nodes, as the one illustrated in Figure 6;
(ii) All nodes guarantee a rate-latency service curve $\beta_{R,T}$ with $R = 100Mb/s$ and $T = 40us$;
(iii) Each node generates leaky-bucket constrained flows with a burst σ and a rate ρ;
(iv) The network has a maximum bottleneck utilisation rate U.

Furthermore, we define various network configurations, where each network configuration is defined with the tuple (σ, ρ, M, U). We vary the network size M or load U of this tuple at a time, to highlight its impact on the derived delay bounds and time analysis.

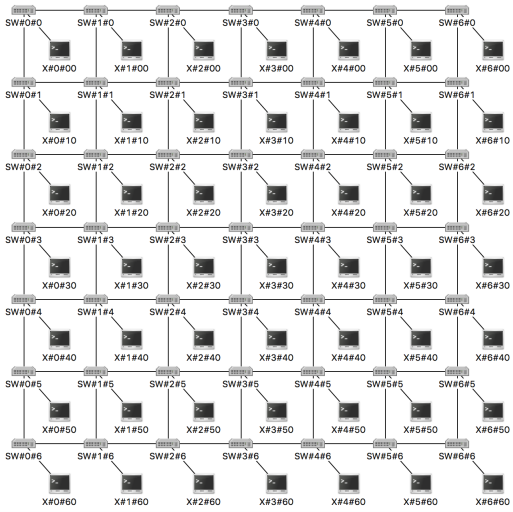

Figure 6: Considered Network topology for the comparative analysis

Figures 7 and 8 show the impact of the network size and load on the delay bounds under the different timing algorithms, respectively. As we can notice, TFA++ implies the lowest delay bounds for both scenarios, in comparison to existing algorithms. This fact shows its high accuracy. Moreover, when increasing the network size, i.e., the number of nodes, or the network load, i.e., the number of flows, after a given threshold value, PMOO leads to worse delay bounds than SFA, since it becomes more and more difficult to find a smart application order of Theorem 2.7 and Theorem 2.5 under these conditions; thus a more pessimistic bounds.

On the other hand, Figures 9 and 10 show the impact of the network size and load on the time analysis under the different timing algorithms, respectively. As illustrated, TFA++ infers almost the same time analysis than the original TFA, which still is the lowest one in comparison to existing algorithms.

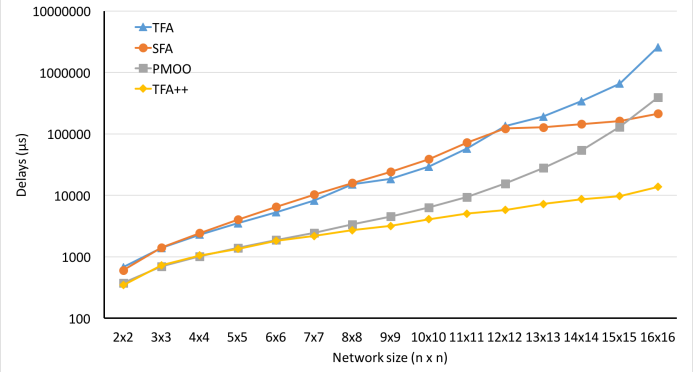

Figure 7: The impact of the network size on the delay bounds for ($\sigma = 400bytes, \rho = 20kbps, M \in [2x2 : 1x1 : 16x16], U = 30\%$)

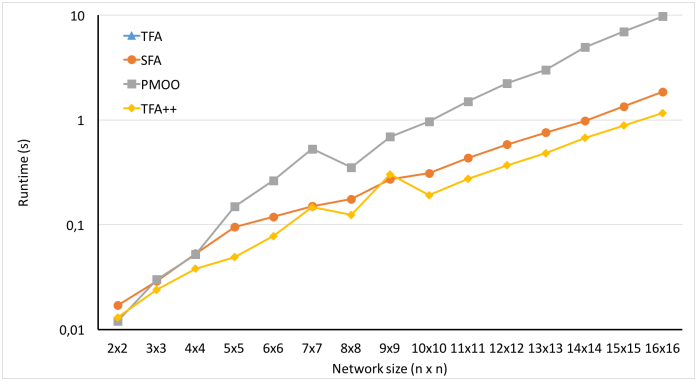

Figure 9: The impact of the network size on the analysis time for ($\sigma = 400bytes, \rho = 20kbps, M \in [2x2 : 1x1 : 16x16], U = 30\%$)

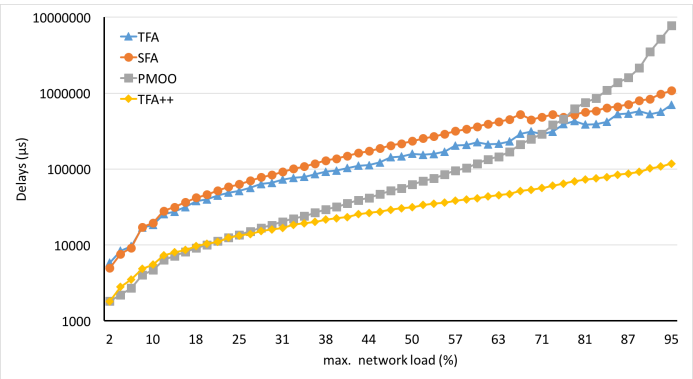

Figure 8: The impact of the network load on the delay bounds for ($\sigma = 1000bytes, \rho = 250kbps, M = 6x6, U \in [2 : 2 : 98]$)

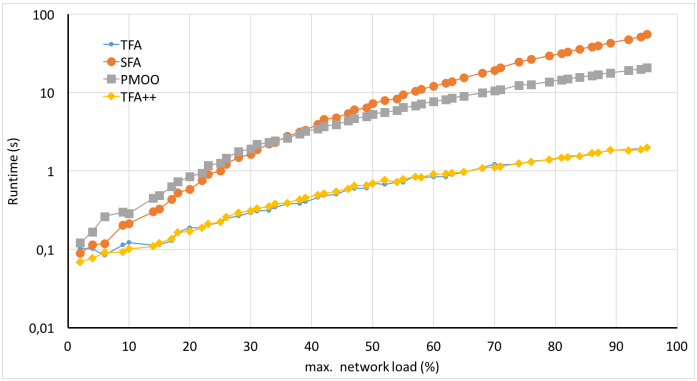

Figure 10: The impact of the network load on the analysis time for ($\sigma = 1000bytes, \rho = 250kbps, M = 6x6, U \in [2 : 2 : 98]$)

In addition, we can notice that the PMOO analysis time becomes the highest when increasing the network size due to the high call number of convolution operator ($\otimes$); whereas when increasing the number of flows, the SFA leads to the highest analysis time due to the high call number of deconvolution operator ($\oslash$).

Hence, these results confirm our first conclusions in Section 3 concerning the pros and cons of the existing algorithms, where the PMOO is considered in general as the most accurate one but also the most complex one. Moreover, they show the performance of our proposed algorithm in terms of high accuracy and low complexity, in comparison to existing solutions.

5.2 Avionics Case Study

We report herein the derived delay bounds under the different timing algorithms for a representative AFDX network of A350, illustrated in Figure 11. This network consists of 7 switches and more than 60 end-systems. The latter generate more than 1100 multicast Virtual Links, inferring more than 8600 flows. The maximum packet lengths are between 64bytes and 1500bytes and the periods are between 4ms and 128ms.

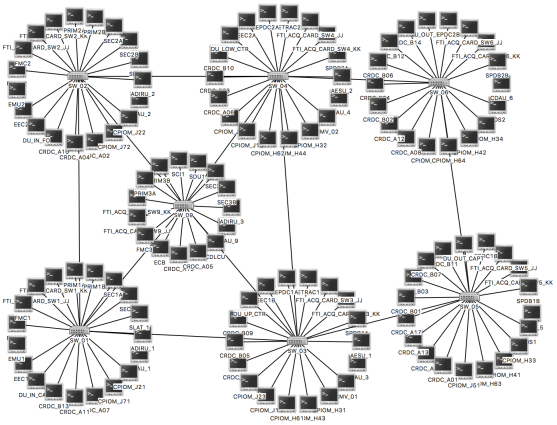

Figure 11: Representative Avionics Case Study

Figure 12 shows the delay bounds for the different flows following the ascending order under the different timing algorithms. As we can notice, both TFA++ and PMOO imply the most accurate delay bounds for most of the flows. However, for this realistic case study, there is no strict order relation between both algorithms. Hence, to derive the tightest delay bounds, we can consider the minimum of the computed delay bounds for each flow using PMOO and TFA++. It is worth noting that the TFA++ still has the lowest time analysis for this avionics case study. This fact shows that our proposal still is a good value for this realistic case study, in comparison to the existing algorithms in terms of accuracy and complexity.

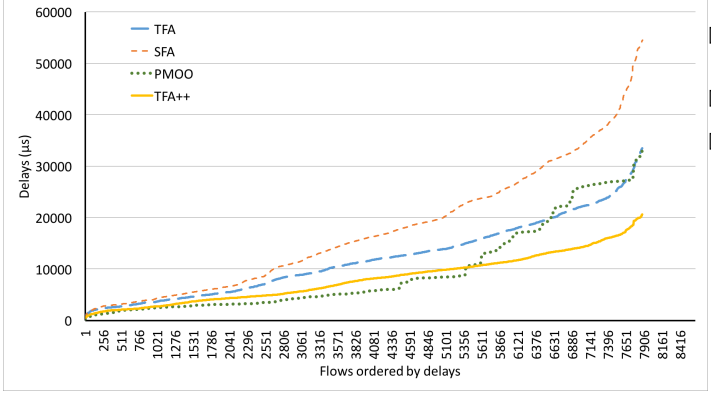

Figure 12: Delay bounds for the avionics case study under the different analysis algorithms

6 CONCLUSIONS

A new timing analysis algorithm based on Network Calculus, called TFA++, has been proposed in this paper to cope with the known accuracy-complexity tradeoffs in the literature. The main idea of TFA++ is to take into account the impact of the finite transmission capacity of the links on the input and output traffic models of each crossed node. Moreover, this algorithm outperforms the main existing solutions, in terms of high accuracy and low complexity for some network configurations.

However, the avionics case has shown that there is no strict order relation between PMOO and TFA in terms of accuracy. This fact encourages us to lead further investigations on the comparative analysis of both algorithms, when considering various network topologies and flow parameters, i.e., the burst, the rate or the path length.

REFERENCES

[1] WoPANets: Worst-Case Performance Analysis of embedded Networks. http://websites.isae.fr/wopanets.
[2] A. Amari and A. Mifdaoui. Worst-case Timing Analysis of Ring Networks with Cyclic Dependencies using Network Calculus. Proceedings of RTCSA, 2017.
[3] A. Bouillard, L. Jouhet, and E. Thierry. Service curves in Network Calculus: dos and don'ts. Technical report, 2009.
[4] A. Bouillard and G. Stea. Exact Worst-case Delay in FIFO-multiplexing Feed-forward Networks. *IEEE/ACM Transactions on Networking*, 2014.
[5] C.-S. Chang. *Performance Guarantees in Communication Networks.* Springer-Verlag, 2000.
[6] R. L. Cruz. A calculus for network delay. I. Network elements in isolation. *Information Theory, IEEE Transactions on*, 37, 1991.
[7] R. L. Cruz. A calculus for network delay. II. Network Analysis. *Information Theory, IEEE Transactions on*, 37, 1991.
[8] J. Grieu. *Analyse et evaluation de techniques de commutation Ethernet pour l'interconnexion de systemes avioniques.* PhD thesis, INP, Toulouse, 2004.
[9] J. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the internet.* Springer-Verlag, 2001.
[10] L. Lenzini, E. Mingozzi, and G. Stea. A Methodology for Computing End-to-end Delay Bounds in FIFO-multiplexing Tandems. *Performance Evaluation*, 2008.
[11] J. Loeser and H. Haertig. Low latency hard real-time communication over switched Ethernet. IEEE, Proceedings of Euromicro Conference on Real-Time Systems, 2004.
[12] S. Perathoner, E. Wandeler, and et al. Influence of Different Abstractions on the Performance Analysis of Distributed Hard Real-Time Systems. *Design Automation for Embedded Systems*, 2009.
[13] Y. Qian, Z. Lu, and W. Dou. Analysis of worst-case delay bounds for on-chip packet-switching networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2010.
[14] J. B. Schmitt and F. A. Zdarsky. The DISCO Network Calculator - A Toolbox for Worst Case Analysis. In Proceedings of the First International Conference on Performance Evaluation Methodologies and Tools, 2006.
[15] J. B. Schmitt, F. A. Zdarsky, and M. Fidler. Delay bounds under arbitrary multiplexing: When network calculus leaves you in the lurch... INFOCOM, 2008.
[16] J. B. Schmitt, F. A. Zdarsky, and I. Martinovic. Improving Performance Bounds in Feed-Forward Networks by Paying Multiplexing Only Once. MMB, 2008.