Breaking vs. Solving: Analysis and Routing of Real-time Networks with Cyclic Dependencies using Network Calculus*

Anaïs Finzi
anais.finzi@tttech.com
TTTech Computertechnik AG

Silviu S. Craciunas
silviu.craciunas@tttech.com
TTTech Computertechnik AG

ABSTRACT

Distributed real-time systems in the aerospace domain require worst-case end-to-end latency analysis methods to provide certification evidence of the correct temporal behavior of critical traffic classes. One such analysis method is the Network Calculus framework. While the Network Calculus analysis is mature enough to be allowed in certification artefacts, it is only applied in networks where there are no cyclic dependencies between communication flows (so-called feed-forward networks). In general topologies, flows can form cyclic dependencies, making it difficult to prove the determinism of a network. There are two approaches to solve this problem: 1) breaking the dependencies in the routing algorithm to study a feed-forward network; 2) solving, i.e., computing the bounds, in the dependency. In this paper, we review the recent improvements of both methods and do a performance analysis of AFDX and TTEthernet networks to compare their impact on the worst-case delay and backlog bounds. Results show that the best method depends on a number of parameters, such as load and dependency length. Using these results, we propose a new routing methodology resulting in the lowest bounds for networks with cyclic dependencies.

KEYWORDS

Formal timing analysis, Cyclic dependencies, Network Calculus, AFDX, TTEthernet, Routing

ACM Reference Format:

Anaïs Finzi and Silviu S. Craciunas. 2019. Breaking vs. Solving: Analysis and Routing of Real-time Networks with Cyclic Dependencies using Network Calculus. In *27th International Conference on Real-Time Networks and Systems (RTNS 2019), November 6–8, 2019, Toulouse, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3356401.3356418

1 INTRODUCTION

Distributed real-time systems, like those found in aerospace systems, require a certification process (i.e. proof of determinism) that proves the correct temporal behavior of critical communication flows in terms of end-to-end latency (delay) and backlog constraints. Due to the increasing communication requirements, Airbus has developed a new standard in 2007, the ARINC 664 specification part 7 [1], known as Avionics Full DupleX (AFDX), to be used as a backbone in its latest aircraft systems (e.g. the A380 and A350). Several technologies, like TTEthernet [29], and TSN [18], have been introduced to address more stringent real-time communication requirements over standard Ethernet.

The worst-case end-to-end delay requirements for these kind of networks have been guaranteed through methods like Network Calculus [13, 16, 17] or the more recent Compositional Performance Analysis [31]. The Network Calculus method [17] is a well-known mathematical framework that uses min-plus algebra in order to derive worst-case bounds for individual communication flow latency and on the backlog in individual output ports. For example, in TTEthernet networks, a rate-constrained (RC) traffic class analysis which takes into account the time-triggered (TT) traffic class has been introduced in [7, 32, 33].

While the Network Calculus analysis is mature enough to be allowed in certification artefacts, it is only applied in feed-forward networks, i.e. network where the paths of interfering communication flows do not form cycles (so-called cyclic dependencies). However, most networks in production systems have the possibility of having cyclic dependencies. The most common mitigation method for the problem of cyclic dependencies in the analysis is to break the dependencies. Breaking the dependencies can be done in the routing process to obtain easy-to-study feed-forward networks. A consequence of this approach is an increase of the end-to-end delay and backlog bounds due to the potential unbalancing of the RC/AFDX traffic necessary to avoid cyclic dependencies. Nonetheless, until now, this has been deemed a better alternative than solving the dependencies, which has an increased complexity and may often lead to pessimistic solutions. However, the introduction of new architectures based on rings [3] has lead to significant improvements in cyclic dependency solving methods. When analyzing the recent results in the Network Calculus theory on cyclic dependencies (e.g. [2], [4]) and results in feed-forward networks (e.g. [8], [22]), it is not clear whether breaking cyclic dependencies in the routing process still provides the lowest delay and backlog bounds or not. Finally, the question is whether the higher real RC/AFDX bounds in feed-forward networks with tight bounds are better or worse than lower real RC/AFDX bounds in cyclic dependency networks with pessimistic bounds. In other words, is it more beneficial for the analysis to solve or to break dependencies in such networks?

In this paper, we address this question in AFDX and TTEthernet networks and propose an analysis of the two methods: 1) cycle breaking in the routing, associated to computations for feed-forward networks; 2) computations within networks containing

*This project has received funding from the Clean Sky 2 Joint Undertaking under the European Union's Horizon 2020 research and innovation programme under grant agreement n° 807081.

RTNS 2019, November 6–8, 2019, Toulouse, France

ACM ISBN 978-1-4503-7223-7/19/11…$15.00
https://doi.org/10.1145/3356401.3356418

cyclic dependencies. In both cases, we conduct a performance analysis based on the Network Calculus framework which considers ring topologies. Hence, this is the first work about the trade-offs between solving and breaking cyclic dependencies in AFDX and TTEthernet networks. We consider four main constraints:

1) the analysis is done in the avionics context, i.e. for the AFDX and TTEthernet standards;

2) the analysis considers general topology networks, i.e., not restricted to ring topologies;

3) the traffic load must be analyzed up to 100%;

4) the runtime complexity of the solution must be preferably linear, or polynomial.

There are two reasons justifying this last constraint. First, to obtain the needed certification in avionics, software must fulfill many criteria. Hence, complex solutions make the certification process more difficult and costlier. Secondly, in networks mixing RC and TT frames, the computation of RC bounds within the routing and scheduling of TT frames is necessary to enforce the RC deadlines (cf. [30]). As a result, low computation time (i.e. linear complexity) is needed to make the process end in a feasible amount of time.

Currently, the AFDX is a fully asynchronous network. However, extensions that add synchronization to AFDX networks have been proposed [6]. Hence, in addition to the AFDX network, we consider a second type of real-time Ethernet network that we study in this work, namely TTEthernet [29]. This will enable us to also study the impact of the Time Triggered higher priority frames within a time synchronized network.

In particular, we study so-called tipping points, i.e., the point where the best performance tips from one method to the other. We consider the impact of the length of cyclic dependencies, the length of the flow path in this cycle, and the impact of higher priority traffic, e.g., TT frames. This enables us to define different areas depending on the best solution. Using this analysis, we are then able to propose a method to select the best variant based on the tipping points and define a new routing approach that outperforms classical methods.

Therefore, our main contributions are: 1) the overview and comparison of the recent results with Network Calculus on dealing with cyclic dependencies in networks with arbitrary topologies, in Section 3; 2) a performance analysis done on rings, to determine the best solution depending on multiple parameters, such as load and cycle length, in Section 4; 3) a new routing method to obtain the lowest Network Calculus bounds, in Section 5.

2 BACKGROUND

Here we give an overview of the Network Calculus framework and present two reference network technologies that we apply our study on, namely AFDX and TTEthernet networks. Finally, we illustrate the challenges of the current A380 and A350 AFDX architecture. The main notations used in this paper are defined in Table 1.

2.1 Network Calculus

The timing analyses detailed in this paper are based on the Network Calculus framework [20]. It is used to compute upper delay and backlog bounds. These bounds depend on the traffic arrival described by the so-called *arrival curve* α, which represents the maximum amount of data that can arrive in any time interval, and on the availability of the crossed node described by the so-called minimum *service curve* β, which represents the minimum amount of data that can be sent in any time interval. The definitions of these curves are detailed below.

$C^n_{I,in}$	sum of the capacities C_{in} of the input links of node n crossed by traffic of class I
MFS_i	Maximum Frame Size of flow i
BAG_i	Bandwidth Allocation Gap of flow i
b^n_i, r_i	burst, rate of the input arrival curve of flow i in node n
R^n_i, T^n_i	rate, initial latency of the minimum service curve offered to flow i in node n
WCD^n_i	Worst-Case Delay of flow i in node n
WCB^n_i	Worst-Case Backlog of flow i in node n
WCD^n_{fwd}	Worst-Case forwarding Delay in node n
BCD^n_{fwd}	Best-Case forwarding Delay in node n
$bound_k$	maximum bound of $bound \in \{delay, backlog\}$ of method $k \in \{solve, break\}$
N, L	number of switches in the ring and flow path length
UC_k	use-case $k \in \{1, 2\}$
$RC^{relative}_{LOW}$	relative load of RC_{LOW} class with regard to the remaining capacity left by RC_{HIGH} class
RC^{prim}	RC load on the primary path, i.e., default path with the solving method
RC^{sec}	RC load on the secondary path, i.e., after using the breaking method

Table 1: Notations

Definition 1 (Arrival Curve). *[20] A function $\alpha(t)$ is an arrival curve for a data flow with an input cumulative function $A(t)$,i.e., the number of bits received until time t, iff: $\forall t, A(t) \leq A \otimes {}^1\alpha(t)$*

Definition 2 (Strict minimum service curve). *[20] The function β is a minimum* strict *service curve for a data flow with an output cumulative function A^*, if for any backlogged period $]s,t]^2$: $A^*(t) - A^*(s) \geq \beta(t-s)$*

To compute the main performance metrics, we need the following results:

Theorem 1 (Performance Bounds). *[20] Consider a flow F constrained by an arrival curve α crossing a system $\mathcal{S}$ that offers a minimum service curve β and a maximum service curve γ. The performance bounds obtained at any time t are:*
Backlog[3]: $\forall\ t:\ q(t) \leq v(\alpha, \beta)$, Delay[4]: $\forall\ t:\ d(t) \leq h(\alpha, \beta)$,
Output arrival curve[5]: $\alpha^*(t) = (\alpha \oslash \beta)(t)$

Theorem 2 (Concatenation-Pay Bursts Only Once). *[20] 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 service curve $\beta_1 \otimes \beta_2$.*

Theorem 3 (Left-over service curve - Non preemptive Static Priority (NP-SP) Multiplexing). *[5] Consider a system with the strict service curve β and m flows crossing it, f_1,f_2,..,f_m. The maximum packet length of f_i is $l_{i,max}$ and f_i is α_i-constrained. The flows are scheduled by the NP-SP policy, where priority of f_i > priority of $f_j \Leftrightarrow i < j$. For each $i \in \{1,..,m\}$, the strict service curve of f_i is given by[6]: $(\beta - \sum_{j<i} \alpha_j - \max_{k \geq i} l_{k,max})_\uparrow$*

[1] $f \otimes g(t) = \inf_{0 \leq s \leq t} \{f(t-s) + g(s)\}$
[2] $]s,t]$ is called backlogged period if $A(\tau) - A^*(\tau) > 0, \forall \tau \in]s,t]$
[3] v: maximal vertical distance
[4] h: maximal horizontal distance
[5] $f \oslash g(t) = \sup_{s \geq 0} \{f(t+s) - g(s)\}$
[6] $g_\uparrow(t) = \left(\sup_{0 \leq s \leq t} g(s)\right)^+$, with $(x)^+ = \max(0, x)$

The traffic contracts are generally enforced using a leaky-bucket shaper, i.e., the traffic flow is (r, b)-constrained where r and b are the maximum rate and burst, and the arrival curve is $\alpha(t) = r \cdot t + b$. A common model of the minimum service curve is the rate-latency curve, defined as $\beta_{R,T}(t) = R \cdot (t - T)^+$, where R is the output transmission capacity, T is the system latency, and $(x)^+$ denotes the maximum between x and 0.

2.2 AFDX and TTEthernet

The AFDX [1] standard manages exchanged data through the Virtual Link (VL) concept. The VL represents a multicast communication from one sender End-System to one or more receiver End-Systems. This concept provides a way to reserve a guaranteed bandwidth for each traffic flow. Each VL is characterized by: (i) BAG (Bandwidth Allocation Gap), ranging in powers of 2 from 1 to 128 milliseconds, which represents the minimal inter-arrival time between two consecutive frames; (ii) MFS (Maximal Frame Size), ranging from 64 to 1518 bytes, which represents the size of the largest frame sent during each BAG. The AFDX standard specifies a Non preemptive Static Priority scheduler based on two priority levels, *LOW* and *HIGH* within nodes. These Rate-Constrained traffic classes are denoted by RC_{HIGH} and RC_{LOW}.

Currently, the backbone network on the A380 contains two independent AFDX networks, a primary and a secondary live-backup, each containing 9 switches forming a partial-mesh network [10], as illustrated in Fig. 1. Within this mesh, we can identify several rings, for example {A,B,C,D}, or the ring using all 9 switches. Within these rings, it is possible for flows to form cyclic dependencies, as illustrated in Fig. 2(a).

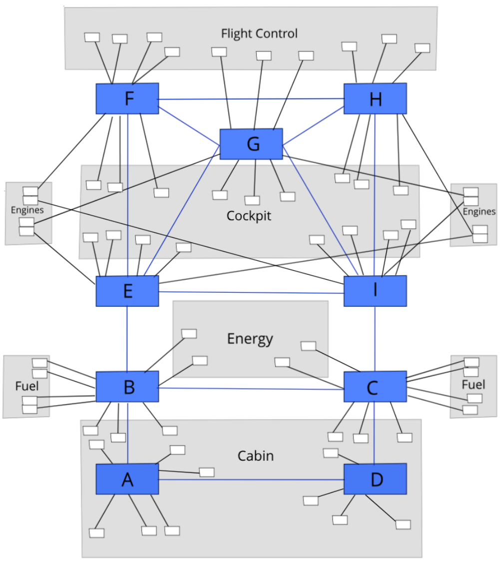


Figure 1: Structure of AFDX on the A380

The TTEthernet [29] standard is based on the use of global time synchronization to send Time Triggered (TT) frames at precise, predefined times (encoded in a local event table that is part of a globally computed schedule) to ensure the lowest contention and delays. Hence, the TT flows are defined by their size, period, and offsets (the times their transmission should start) in each output port. The synchronization uses specific VLs and flows called Protocol Control Frames (PCF) to exchange data needed to compute and communicate the global time to all participants. These PCF flows have the highest priority in TTEthernet networks, the next priority is used by the TT flows, the two next priorities are used by the AFDX RC_{HIGH} and RC_{LOW} traffic, respectively, while the 4 lowest priorities are reserved for Best-Effort (BE) traffic (cf. Fig. 3).

3 EXISTING SOLUTIONS AND THEIR LIMITATIONS

We present existing solutions for breaking and solving cycles. To avoid cyclic dependencies in Network Calculus equations, reshaping methods [19] or partitioning service policies (e.g. DRR, WRR, AVB) can be used. Shifting the priority level of a flow can also be a solution. In this paper, we focus on standard AFDX and TTEthernet networks where the priority of the flow is set as input constraint. Then, breaking dependencies is done by re-routing.

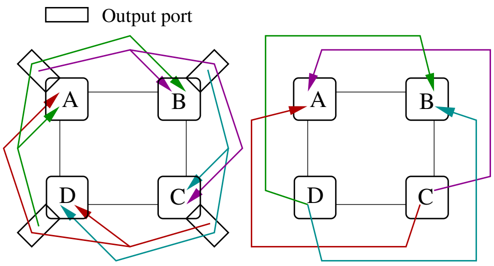
3.1 Breaking cyclic dependencies

Routing: although there are many methods for breaking cyclic dependencies, we focus here on methods fulfilling three main criteria: the method must be 1) deterministic (dynamic routing is not acceptable in avionics); 2) applicable to general topologies, (2D-mesh specific methods, such as XY routing, are out of scope); 3) have a polynomial complexity and require no modifications to an existing network (methods using virtual channels are out of scope). Among the remaining solutions, for example $Up^*/Down^*$ Routing [26], or L-Turn [23], many are based on spanning trees. Consequently, the routing can be unbalanced, with more congestion at the root of the tree than in the other nodes.Hence, we have discarded solutions based on spanning trees. To solve this unbalanced routing, other solutions, based on the prohibition of turns have been developed, such as turn-prohibition [27] and segment-based [21] routing. A drawback is the inability to guarantee the shortest path.

Finally, from our overview, we have selected the Turn-prohibition routing. It is a simple method, with good performances and contrary to Segment-based routing, the weight of the links can be taken into account to improve performance. To obtain the final routing of the different flows, we use the Turnnet Algorithm [14] in order to transform a graph of node=output port and edge=link, into a new graph with node=link and edge=turn. The forbidden turns can now be removed from the graph. Finally, we apply the well-known Dijkstra algorithm on this new graph in order to find routes.

Network Calculus in feed-forward networks:

in this paper, we used the Serialization Effect Paradigm (SEP) proposed in [16], illustrated in [8], and recently compared with other solutions in [22], under the name of Total Flow Analysis++ (TFA++). The idea is to take into account the link capacity of the input ports in order to obtain a more accurate input arrival curve of the flows. For example, a flow arriving with a maximum burst b and rate r, from a link with a capacity C_{in}, has an input arrival curve $\alpha(t) = min(C_{in} \cdot t, r \cdot t + b)$. With this tighter $\alpha(t)$, the bounds are computed in each output port. For a flow of interest, the end-to-end delay is the sum of the delays in each output port along the



(a) Cyclic dependency (b) Impossible to use breaking method

Figure 2: 4-switches ring traffic examples

route of the flow. Our current implementation of SEP/TFA++ uses piecewise linear concave functions, which has a linear complexity in the number of segments. More complex arrival curve such as staircase curves (subset of Ultimately Pseudo Periodic [8, 9]) can improve the results but at the expense of higher complexity and thus a lower execution time (between 1.4 and 10 times slower).

The analysis in [22] compared SEP/TFA++ to three methods:

1) the Total Flow analysis [24], i.e., iterative computation of upper bounds in each crossed node for the aggregate flow, followed by the computation of the sum of the delays in the crossed nodes. It is important to note that in [22], the individual flows are considered, rather than the aggregate flows;

2) Separated Flow Analysis [24], i.e., concatenation of left-over service curves guaranteed to the flow of interest, based on Th. 2;

3) Pay Multiplex Only Once (PMOO) [25], i.e., taking into account the flow serialization along the path to compute tighter bounds.

Results showed the delay bounds and computation times with SEP/TFA++ are largely reduced compared to the other methods [22].

Discussion: the use of turn-prohibition routing associated to SEP/TFA++ requires simple routing adaptations and is based on tight bounds. However, such a solution has a few drawbacks. First, it can increase the load on some of the links, resulting in additional delays. Secondly, cycles cannot always be broken, especially with redundant flows. Let us take for example, the case of 4 switches A, B, C, and D, forming a ring such as the one in Fig. 1. We consider the following redundant flows on the primary network: A → B → C and A → D → C, B → A → D and B → C → D, C → B → A and C → D → A, D → A → B and D → C → B. These flows form two cyclic dependencies. One of them is illustrated in Fig. 2(a). Different flows are shown in Fig. 2(b), where we can see there are only two disjoint paths for a redundant flow, so all the turns are used. Hence, breaking the cyclic dependency is impossible. A second case where breaking the cyclic dependency could be impossible can happen if the load of the RC traffic becomes higher than the available capacity due to the unbalance created by the cycle breaking.

3.2 Solving cyclic dependencies method

A second solution consists in keeping paths with cyclic dependencies and obtaining the delay and backlog bounds using specific Network Calculus methods. Four methods have been developed:

1) Backlog-based Method (BbM) [20]: for a traffic class I, we compute a single maximum bound for the backlog in any output port of the cyclic dependency, denoted WCB_I for Worst-Case Backlog. Knowing the minimum service curve offered to the considered traffic for the aggregate flow of class I, $\beta_I^k(t) = R_I^k \cdot (t - T_I^k)^+$, (with T_I^k the initial latency and R_I^k the transmission capacity), the Worst-Case Delay (WCD) within a node k for a flow of class I is the processing time of WCB_I with R_I^k: $WCD_I^k = \frac{WCB_I}{R_I^k}$.

2) Time Stopping Method (TSM) [12]: the main idea is to find the equations linking the bursts and the delays in each output port, for each flow, then use matrices to solve these equations. We denote r_i the rate of a flow i of class I, b_i^k the burst of flow i entering a node k, b_i^0 the burst entering the cycle dependency, and WCB_I^k the worst-case delay of the flows of class I in node k. First, the burst of a flow i exiting a node k is equal to $b_i^k + r_i \cdot WCD_I^k$. Hence, the output burst of a flow i in a node n is:

$$b_i^n = b_i^0 + r_i \cdot \sum_{p \in prec(n,i)} WCD_I^p \tag{1}$$

with $prec(n, i)$ the set of nodes preceding n in the cyclic dependency in the path of flow i. Secondly, the worst-case delay of a flow of class I in a node k is the sum of the initial latency of the minimum service curve, T_I^k and the processing time of all the bursts of the flows $j \in I$ crossing k:

$$WCD_I^k = \frac{\sum_{j \ni k} b_j^k}{R_I^k} + T_I^k \tag{2}$$

Finally, with Eq. (1) and Eq. (2) we can define a matrix system:

$$\begin{cases} D = A_1 \cdot B + C_1 \\ B = A_2 \cdot D + C_2 \end{cases} \tag{3}$$

with D the vector of WCD_I^k, B the vector of propagated bursts b_i^n, C_1 and C_2 are constant vectors. The resolution of this system gives: $D = (I - A_1 \cdot A_2)^{-1} \cdot C_3$, with $C_3 = A_1 \cdot C_2 + C_1$ and I the identity matrix. This system admits a positive solution only if $(I - A_1 \cdot A_2)$ has a strictly positive determinant.

3) Pay Multiplexing Only at Convergence point (PMOC) [2]: consider the flow serialization along the path of the flow of interest by paying the bursts only at convergence points. Similarly to TSM, equations are developed to obtain a system of matrix with two unknown vectors, like Eq. (3).

The difference is that they consider individual flows. As a result, solving is quite heavy with large matrices. For instance, if we consider the current A380 architecture with 9 switches, there are 26 ports linking the switches, and over a thousand VLs. If we consider that the length of the path of the VLs is only two, this gives matrices D and B of size 2 000 × 1 to represent the unknown propagated bursts and delay bounds for each flow in each crossed output port. Finally, the matrices A_1 and A_2 describing the equations would have a size of 2 000 × 2 000. With TSM, the matrix size would only be 26×52 for A_1 and 52×26 for A_2. Hence, the PMOC method is more complex and requires a larger amount of computation power than TSM. Additionally, the authors do not provide a way of computing the backlog in each output port.

4) Tree decomposition [4]: the stability of networks with cyclic dependencies is analyzed and a method to decompose the cycles into trees is proposed. The trees are then analyzed using feed-forward methods. It is shown that different decompositions result in different stability and bounds. Hence, a optimization problem is introduced to find the best results. According to [4], the computational time of the algorithm is polynomial.

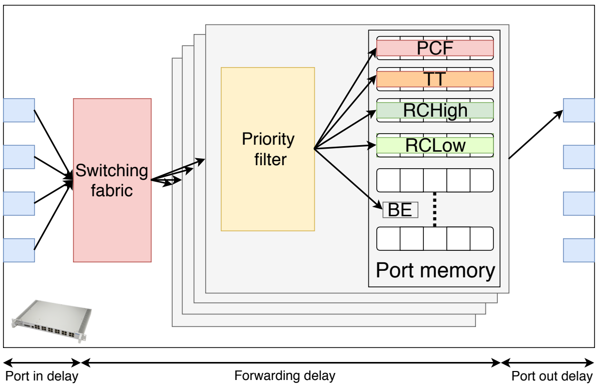


Figure 3: Considered switch architecture [15]

Discussion: The first three methods have been compared in [2]. Firstly, the main advantages of BbM is the fact that bounds can be computed up to a 100% load with good complexity. As a result, the down-side is its pessimism. Secondly, concerning TSM, the method also has a polynomial complexity, and bounds are tighter until they diverge. As a consequence, the bounds cannot be computed after the divergence point. Thirdly, PMOC is more complex, with tighter bounds, but does not provide a backlog computation method and also has a divergence point.

The last method, i.e. tree decomposition, is not compared to the other solutions (PMOC, TSM and BbM) in [4]. Moreover, the timing analysis requires an additional step to decompose the network into trees, using optimization methods of polynomial complexity. Hence, we discard this method.

Two of our constraints are to obtain a method computing delay and backlog bounds up to a 100% load, with good complexity. As a results, even though PMOC has better tightness, we have selected the first two methods: TSM and BbM. The analysis will be done using both methods and we will keep the minimum value.

4 PERFORMANCE ANALYSIS

In this section, we do a performance analysis to compare the two selected solutions for breaking or solving cyclic dependencies. First, we present preliminary assumptions, e.g., the model of the flow, the AFDX and TTEthernet switch and network models, and delay and backlog computations. Then, after presenting our case study, we do a comparison of the two methods for AFDX and TTEthernet.

4.1 Preliminaries and assumptions

Switch and End-System model: we consider the TTEthernet switch architecture described in Fig. 3. If the switch is AFDX only, the two highest priorities will be RC_{HIGH} and RC_{LOW}, respectively. The input port delay is the amount of time needed for a frame i to fully arrive at a rate C_{in}: $\frac{MFS_i}{C_{in}}$. We consider the delay in the switch starts after the frame has been fully received. The forwarding process is defined by a minimum (best-case) and maximum (worst-case) delay, denoted WCD^n_{fwd} and BCD^n_{fwd}, in a switch or end-system $n \in \{sw, es\}$.

AFDX and TTEthernet output ports: for TTEthernet, the impact of the TT traffic on RC traffic is computed using the input arrival curves proposed in [32]. Then, in both cases, the service curves are computed using Th. 3.

Traffic model: to compute the delay and backlog bounds within each node (output port, switch sw or end-system es), we use Th. 1 under the following assumptions:

(i) leaky-bucket arrival curves for the traffic flows at the input of node n. For a flow i, we define the Maximum Frame Size MFS_i and the Bandwidth Allocation Gap BAG_i (the period and generally also the deadline), and the initial jitter J_i. The initial arrival curve sent by the traffic source is $\alpha_I(t) = \sum_{i \in I}(\frac{MFS_i}{BAG_i} \cdot (t + J_i) + MFS_i)$. For each class I, the aggregate traffic has an input arrival curve in the node $n \in \{es, sw\}$: $\alpha^n_I(t) = \min(C^n_{I,in} \cdot t, r_I \cdot t + b^n_I)$, where the maximum input rate $C^n_{I,in}$ is the sum of the capacities C_{in} of the input links of node n crossed by traffic of class I. We considered that in a node es generating class-I traffic, $C^{es}_{I,in} = +\infty$;

(ii) the offered service curve by node n to the traffic class I is a rate-latency curve: $\beta^n_I(t) = R^n_I \cdot (t - T^n_I)^+$.

Backlog computation: we consider that after arriving in the input port, the frame is stored in a memory until it is ready to be transmitted. As a consequence, to compute the maximum backlog, we must take into account the arrival curve entering the node $n \in \{es, sw\}$, i.e. $\alpha^n_I(t) = \min(C^n_{I,in} \cdot t, r_I \cdot t + b^n_I)$, and the concatenation (Th. 2) of the minimum service curves of the forwarding process and output port (R^{port}_I and T^{port}_I computed with Th. 3), i.e.:

$$\beta^n_I(t) = R^{port}_I \cdot (t - T^{port}_I - WCD^n_{fwd})^+ \quad (4)$$

Delay bound computation with TSM: to compute the delays and bursts we must take into account the full switch or end-system. Hence, in Eq. (2), we use $\beta^n_I(t)$ described in Eq. (4).

Delay bound computation with SEP/TFA++: with this method, we can use BCD^n_{fwd} to obtain a tighter input arrival curve in the output port. The input arrival curve of an aggregate flow I in the output port $port$ in a node $n \in \{es, sw\}$ is:

$\alpha^{port}_I(t) = \min\left(C^n_{I,in} \cdot (t + \delta^n_{fwd}), r_I \cdot (t + \delta^n_{fwd}) + b^n_I\right)$, with $\delta^n_{fwd} = WCD^n_{fwd} - BCD^n_{fwd}$, and b^n_I the burst entering the node. Then, according to Th. 1, we can compute the worst-case delay bound as the maximum horizontal distance between $\alpha^{port}_I(t)$ and $\beta^{port}_I(t) = R^{port}_I \cdot (t - T^{port}_I)^+$. The delay in the node n is then: $WCD^n_I = WCD^{port}_I + WCD^n_{fwd}$.

End-to-end delay bounds: finally, the end-to-end delay of a flow is obtained by summing the delays in the end-systems, input ports, switches and links along the path of the flow.

4.2 Case Study

When considering a network with a general topology and cyclic dependencies, the areas of interest are the rings formed by the dependent flows. Outside these rings, the sub-networks are feed-forward networks and the delays are computed as explained in Section 3.1, regardless of the method selected to solve the cyclic dependencies.

Hence, in this case study, we consider rings made of N 100-Mbps switches, each connected to End-Systems (as illustrated in Fig. 4 for N=6).

The best-case (BCD^n_{fwd}) and worst-case (WCD^n_{fwd}) forwarding delays are described in Table 2. The forwarding delays of an End-System refers to the delay between the host and an output port.

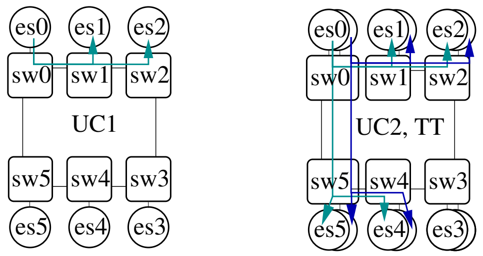


Figure 4: Illustration of 1 VL of each end-system connected to switch $sw0$, with N=6 and L=2, for use-cases UC_1, UC_2 and TT traffic.

Node n	WCD^n_{fwd} (s)	BCD^n_{fwd} (s)
End System	$1.5 \cdot 10^{-6}$	$1.1 \cdot 10^{-6}$
Switch	$12.5 \cdot 10^{-6}$	$10.3 \cdot 10^{-6}$

Table 2: Forwarding delays

The forwarding delays in a Switch refers to the delay between an input port and an output port.

We consider two uses-cases for the RC traffic flows:

- Use-Case 1 (UC_1): each switch is linked to one end-system. Each end-system sends four identical VLs, each multi-casting to 2 destinations, i.e., the end-systems connected to its $(L-1)^{th}$ and L^{th} neighboring switches clockwise (see Fig. 4 with N=6 and L=2);
- Use-Case 2 (UC_2): each switch is linked to two end-systems. Each end-system sends two identical VLs, each multi-casting to 4 destinations, i.e., end-systems connected to its L-1^{th} and L^{th} neighboring switches both clockwise and anticlockwise (see Fig. 4 with N=6 and L=2).

These two use-cases enable us to assess the difference between: i) when the secondary paths (i.e., the paths used due to the cycle breaking) are empty (i.e. UC_1) or ii) when they already have a load (i.e. UC_2).

Varying the BAG of a flow only varies the rate of the arrival curve, whereas by varying the MFS, we vary both the rate and the burst.

The RC traffic has $J_{RFC} = 0$, $BAG_{RC} = 1$ms and we vary the MFS from 100 bytes to 1500 bytes to vary the load of the RC load from 6% to 96%. For instance, with eight VLs in an output port (both use-cases with L=2), and MFS_{RC}=500 bytes, the RC load in the ring is $load_{RC} = 8 \cdot \frac{MFS_{RC}}{BAG_{RC}} = 32$ Mbps or 32% of the 100-Mbps capacity.

In the case of TTEthernet networks, the routes of PCF flows depend on the network topology and assigned synchronization roles but do not vary at runtime.

For the TT traffic, we consider that each switch is linked to two end-systems. The first (resp. second) end-system sends one VL, multi-casting to 2 destinations, i.e., end-systems connected to its L-1^{th} and L^{th} neighboring switches clockwise (resp. anticlockwise), as illustrated in Fig. 4. The TT traffic has a period of 1ms and is scheduled with the shuffling integration method [29], using the SMT-based approach described in [11, 28]. We vary the MFS from 100 bytes to 1500 bytes to vary the TT load from 1% to 24%. The TT schedule in each case is computed offline and cannot be modified at runtime. Please note that generally in aerospace projects, once the TT schedule for the most critical traffic has been computed, it cannot be modified without additional certification effort.

Finally, we consider best-effort traffic with a MFS of 1518 bytes.

The parameters used throughout the performance analysis are derived from the current AFDX design illustrated in Fig. 1, with its N=9 switches, cluster connections, and its load of 30% on only one class. Then, these parameters are extended to analyze possible future configurations by doing a sensitivity analysis and by considering an increasing number of classes.

4.3 AFDX network

We start by studying a network with only one RC class, as is currently the case in the A380 and A350. We notice that L=4 is the maximum path length between switches on opposite parts of the network (e.g. A and H in Fig.1). A further analysis of the current network shows that end-systems inside a cluster, e.g., cockpit or cabin, are always at most 2 switches from each other. Hence, we will mainly use L=4 and L=2 for our experiments. We finish by considering two RC classes to study the impact of a higher priority. We compare the two methods after studying them independently.

1) Impact of RC load on solving methods: we have implemented both TSM and BbM and computed the worst-case delay and backlog. The resulting bounds when varying the RC load are presented in Fig. 5, with a logarithmic scale for both the delays and backlogs. Results show the backlog and delay bounds behave in a similar way, with the TSM bound below the BbM bound until TSM diverges, at an RC load of 65%. On the contrary, BbM steadily increases until 100% RC load. Hence, these methods are complementary, leading us to take the minimum of TSM and BbM bounds.

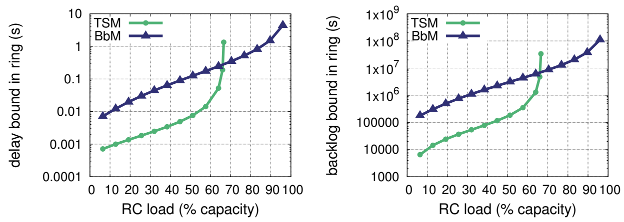


Figure 5: TSM vs BbM for UC_1, N=8 and L=4

Next, we vary the length of the ring N from 8 to 24. The results in Fig. 6 show that in the TSM part of the curve, i.e. below an RC load of 65%, neither the delays nor the backlogs are impacted by the variations of N. This is because the traffic in each output port does not vary when N varies, as we add an end-system for each additional switch, keeping the RC load constant. But, with BbM, the computation of the backlog takes into account the total load of the ring. Thus the delay and backlog bounds increase with N.

2) Impact of RC load on breaking methods: contrary to the solving method, with the breaking method, the delay and backlog bounds increase with both RC load and N, without diverging, as illustrated in Fig. 7. This is because the secondary path, i.e. the path taken to avoid the prohibited turn, also increases with N.

Now that we have studied the behavior of both methods independently, we can now compare them.

3) Comparing the two methods: in Fig. 8, we compare the delay and backlog bounds for L=4 and N=18. We can see there is

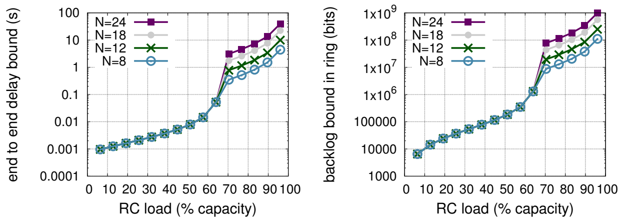


Figure 6: Solving method: varying RC load, with UC_1 and L=4

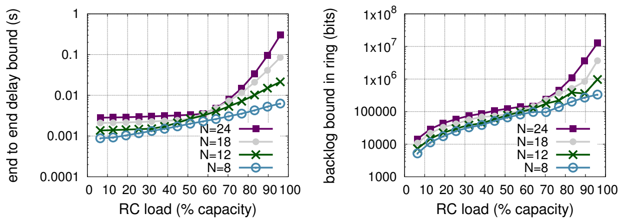


Figure 7: Breaking method: varying RC load, with UC_1 and L=4

no strict order between the methods: for an RC load below 28%, solving has the lower bounds for both the delay and backlog. For an RC load over 28%, breaking the dependency is better due to the divergence of TSM and pessimism of BbM. We call the point where both methods have the same bounds the *tipping point*.

The importance of the tipping point is illustrated in Fig. 9. The delay and backlog bounds computed respectively with breaking and solving are denoted $bound_k$, with $bound \in \{delay, backlog\}$, and $k \in \{solve, break\}$.

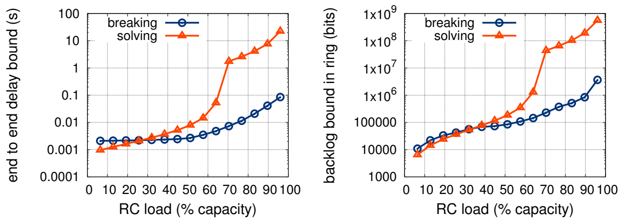


Figure 8: Comparing methods: varying RC load, with UC_1, N=18 and L=4

The improvement of selecting the best method is:

$$\frac{\max(bound_{solve}, bound_{break}) - \min(bound_{solve}, bound_{break})}{\max(bound_{solve}, bound_{break})}$$

We can see that the selection of the best solution is important and can carry sizable bound improvements.

For instance, at RC=20%, the delay (resp. backlog) improves by 28% (resp. 20%) if the solving method is selected over the breaking method. Hence, a good characterization of the tipping point (where the improvement is null in Fig. 9) will help select the best method and largely improve the bound computation. For N=18, the tipping point is roughly the same for the delay and backlog, but this is not always the case. Nevertheless, for the remainder of the paper, we will focus mainly on the delay bounds.

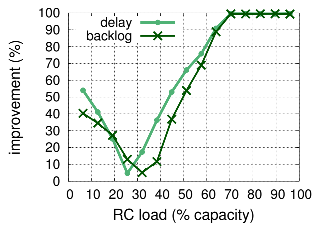


Figure 9: Improvement when selecting the best method compared to the worst method, with UC_1, N=18 and L=4

To characterize the tipping point, we will vary several parameters sequentially: the ring length, the path length and the RC load on the secondary path.

Ring length variation

We varied the size of the ring between N=4 and N=24 to compute this tipping point and represent the areas where solving or breaking is better, as illustrated in Fig. 10(a). Results show that the tipping point increases with the ring length N, e.g., tipping point= 30% at N=8, and tipping point= 100% at N=24. This is due to the fact that when N increases, the delay bound remains constant with TSM, and increases with the breaking method, as illustrated in Fig. 5 and Fig. 7.

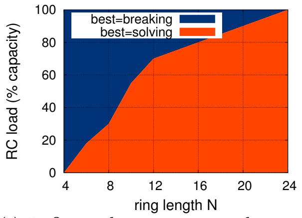
Path length variation

A second parameter influencing the tipping point is the path length L, as illustrated in Fig. 10(b). For L=2, we find the results of Fig. 10(a). Over the tipping point function is the area where breaking is better, under the tipping point function is the area where solving is better. Hence, when L increases, we can see a decrease of the tipping point. For instance, at N=24, the tipping point is at 100% for L=2, 35% for L=4, and 15% for L=6. The reason is that when L increases, the difference of path length between the primary path (with the cyclic dependency) and the secondary path (without cyclic dependency) decreases. However, we also noticed that for the same path difference, we still do not obtain the same tipping point. For instance, for (N=10, L=2) the tipping point is 55%; for (N=12, L=4), the tipping point is 20%. We identify here an impact of the pessimism of the solving method which increases with L and N, i.e., solving is more pessimistic at (N=10, L=2) than (N=12, L=4). This results in poorer performance compared to the breaking method and thus leads to a lower tipping point.

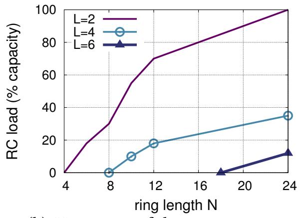
RC load variation on secondary path

With the first use-case UC_1, all the flows turn clockwise, unless they are broken by the turn-prohibition routing method. As a result, the anti-clockwise output ports (for instance A to D in Fig. 2(a)) are empty in the cyclic-dependent network. Hence, when a flow is constrained to turn anti-clockwise, the contention in the output ports is low and there is a low risk of overflow in the output port. In the second use-case UC_2, we consider that all ports are similarly loaded. As a result, when a flow is constrained to turn anti-clockwise, it encounters more contention which increases its delay, and risks overflowing the output ports. This is illustrated in Fig. 10(c), with the RC load on the secondary path in use-case 2 equals the load on the primary path before breaking the dependencies. In use-case 1, the load is null in the secondary path before breaking the dependencies. The test cases depicted in Fig. 10(c) are detailed in Table 3.

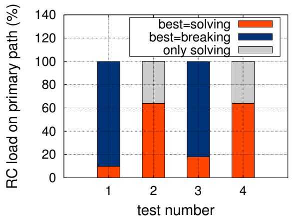
We can see that the tipping point increases in the presence of RC load on the secondary path for both N=6 and N=8. A third area also appears, where breaking is not possible due to the overloading of the output ports, e.g., over 65%. Additionally, we notice that the only-solving area is not affected by N (see test 2 and 4), since the load on the secondary path is not affected by the variation of N.



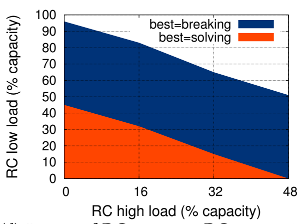
(a) Defining the tipping point between best methods, with L=2



(b) Variations of the tipping point



(c) Impact of the secondary path load with L=2



(d) Impact of RC_{HIGH} on RC_{LOW}, with UC_1, N=10 and L=4

Figure 10: Tipping point and impact analysis

test number	use-case	N	L
1	UC_1	6	2
2	UC_2	6	2
3	UC_1	8	2
4	UC_2	8	2

Table 3: Test cases for Fig. 10(c)

Hence, we can see that in the use-case UC_2, L=2 and N=8, there is no area where breaking should be used. Interestingly, the current AFDX on the A380 has a maximum load of 30%, with the possibility of forming a ring of length N=8 and paths of length L=2. Consequently, according to Fig. 10(c), the best solution is to solve the dependencies rather than break them.

Now that we have presented the results of the tipping point for one RC class, we will study the impact of higher priorities. First, we consider two RC priorities in an AFDX network, then we study the impact of TT flows within a TTEthernet network.

Impact of RC_{HIGH} on RC_{LOW}

We now consider two RC traffic classes and use-case UC_1 for both RC_{LOW} and RC_{HIGH}, with only one VL sent per end-system for RC_{HIGH} (resp. two VLs for RC_{LOW}). Consequently, we vary RC_{LOW} from 6% to 96%, and RC_{HIGH} from 3% to 48%. The results, illustrated in Fig. 10(d), show that the tipping point decreases when RC_{HIGH} increases.

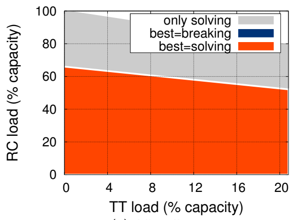
For example, the tipping point is at 32% for $RC_{HIGH} = 16\%$ and 15% for $RC_{HIGH} = 32\%$. This is due to the fact that by increasing the load of RC_{HIGH}, we increase the relative load of RC_{LOW}, i.e., the load of RC with regard to the remaining available capacity left by RC_{HIGH} (instead of the full link capacity). We saw in Fig. 8 that the solving method is more pessimistic than the breaking method for higher value of RC load. Hence, increasing the load of RC_{HIGH} decreases the tipping point due to the pessimism of the solving method.

4.4 TTEthernet network

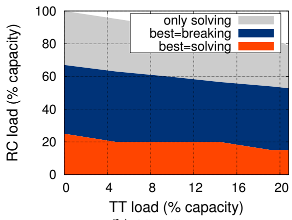
We finish our performance analysis by considering general TTEthernet networks. In particular, we study the impact of TT flows on RC_{HIGH} traffic.

With TT frames going clockwise and anticlockwise, all the ports in the ring receive the same amount of TT traffic. Additionally, TT routing is not concerned by the turn-prohibition algorithm. Concerning RC traffic, we consider use-case UC_2, i.e., the RC load is the same in all output ports in the ring with the cyclic dependencies.

The results are illustrated in Fig. 11. We can see that for close values of N and L, i.e (N=8, L=2) and (N=9, L=4), we obtain very different results.



(a) N=8 L=2



(b) N=9 L=4

Figure 11: Impact of TT on RC_{HIGH}, with UC_2

With (N=8, L=2) in Fig. 11(a), there is no area where breaking is better. On the contrary, with (N=9, L=4) in Fig. 11(b), the area where breaking is the best solution is large: between 20% and 60% for a TT load of 8%, and between 15% and 50% for a TT load of 20%. The first reason is of course the difference of path length when breaking the dependencies, with (N=8, L=2), the path length difference is 5 hops, whereas with (N=9, L=4), the path length difference is 1 hop. The second reason is due to the Time Stopping Method matrices. When L=2, the matrix $(I - A_1 \cdot A2)$ is always invertible and the delay does not diverge. With L=4, $(I - A_1 \cdot A2)$ is not always invertible and the RC delay diverges when the RC load increases, resulting in better delays with the breaking method.

With (N=9, L=4), we notice that the tipping point decreases when the TT load increases, i.e. when the RC relative load increases. This is caused by the pessimism of the solving method which increases with the RC relative load.

In both Figures 11(a) and 11(b) , the lower limit of the only-solving area is due to the overloading of the links when the dependencies are broken, e.g., at TT load=10% and RC load equals 60%, the total traffic on the most loaded output port is 100%, with 60% due to flows on unmodified routes, 30% due to flows on routes modified by turn prohibition and 10% due to the TT frames.

In Fig. 12, we have selected TT load=20%. We have represented the variations of the delay and backlog bounds computed using the solving method, compared to using the breaking method, when breaking is possible, i.e., RC load lower than 45%:

$$\text{relative variation} = \frac{bound_{solve} - bound_{break}}{bound_{break}}$$

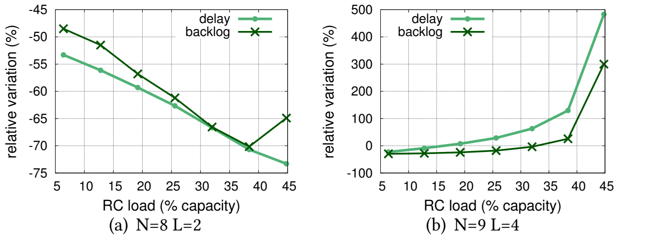


Figure 12: Delay and backlog variations with solving compared to breaking, with UC_2 and TT load = 20%

As expected, in Fig. 12(a), selecting the solving method over the breaking method improve both the delays and backlog, e.g., the delay and backlog are reduced between by 65% for a RC load of 32%. In Fig. 12(b), under an RC load of 15%, the delay bound with the solving method is lower than the delay with the solving method, and over 15%, breaking is better, as illustrated also in Fig. 11(b).

It is interesting to note that selecting the solving solution can have large rewards in terms of delay bounds, e.g., a reduction of 70% at an RC load of 40% with (N=9, L=4). However, it can also be very costly, e.g., an increase of 200% at an RC load of 45% with (N=8, L=2). We obtain similar results for the backlog, with a reduction of up to 70% of the backlog bound with the solving method. Hence, simply selecting one method or the other is not a good solution.

4.5 Discussion

In this performance analysis, we have shown the importance of selecting the best solution between solving or breaking cyclic dependencies, especially within rings. In particular in Fig. 9 and Fig. 12, we showed that selecting the best solution can largely reduce the delay (resp. backlog) bounds, e.g., up to over 75% (resp. 70%), and that selecting one solution over the other can be very costly, with significant bound increases, e.g., up to 500%.

To select the best solution, we have done a sensitivity analysis of the delay bounds to identify the so-call tipping point, i.e., when the two methods have identical bounds.

From our analysis, we can draw a few conclusions about the tipping point:

- *variation of the ring length* (see Fig. 10(a)): we have shown that the tipping point increases when the ring length increases, due the higher length of the secondary path which increases the delay of the breaking method;
- *variation of the path length* (see Fig. 10(b)): results have shown that increasing the path length decreases the tipping point, due to the lower path length difference between the primary path and secondary path, which decreases the delay of the breaking method;
- *variation of RC load on secondary path* (see Fig. 10(c)): our studies have shown that the tipping point increases when the load on the secondary path increases, due to the added contention which increases the delay of the breaking method;
- *variation of the RC_{HIGH} load on primary path* (see Fig. 10(d)): we have shown that the tipping point decreases when RC_{HIGH} load increases, due to both the reduction of the remaining available capacity to RC_{LOW} and the pessimism of the solving method;
- *variation of the TT load on both primary and secondary paths* (see Fig. 11): results have shown the tipping point decreases when the TT load increases, due to the pessimism of the solving method which increases with the relative load.

Concerning the limit over which breaking is not possible, we have shown in Fig. 10(c) and Fig. 11 that increasing the contention on the secondary path decreases the possibility of using the breaking method due to output port overloading.

Additionally, for the current A380 AFDX network, we have show in Fig. 10(c) that under our hypothesis, at the current maximum RC load, i.e., 30%, the best solution is:

- for UC_1, to break the dependencies;
- for UC_2, to solve the dependencies.

This analysis has been done on rings, but even in general topology re-routing would increase the load on a secondary path (to a smaller extend), which would lead to increased contention, increased delays, and sometimes would lead to reaching the maximum load of a link.

Hence, we have shown that always selecting one method over the other is not a viable solution, in particular within rings. Consequently, in the next section we propose a methodology to select the best solution to resolve the cyclic dependency issue.

5 TIPPING POINT METHOD

With our performance analysis, we have shown that the best method is based on several criteria (inputs), i.e., ring length, path length, output port load for each class. For example, for an AFDX network with 2 RC classes, the inputs are: 1) ring length; 2) path length; 3) output port load for RC_{HIGH} on both primary and secondary paths; 4) output port load for RC_{LOW} on both primary and secondary paths. With the flows used for UC_1 and UC_2, all the output ports in the same direction (i.e. clockwise or anti-clockwise) are identically loaded. Hence, we only need to consider one value each for the loads. For the general case, each output port with traffic must be considered.

The main idea of this method is to populate a database with inputs associated to the corresponding best method, to enable the construction of the tipping point and corresponding best method areas.

5.1 Database description

This database can be described by a two-dimensional array. There is one column (i.e. data field) for each input, plus one for the best solution, and one optional to describe the input set id. Each row corresponds to a given input set characterized by the inputs and best solution. In the case of the best solution field, the possible entries are solving, breaking, only-solving, and tipping point. For instance, a few input sets are presented in Table 4. We consider only one class, so there are 5 mandatory fields, i.e., N, L, RC loads and best solution, plus the input set id field.

5.2 Method steps

There are three steps in our method: input computation, identification of best solution in database and updating database. They are detailed in this section.

input set id	N	L	RC^{prim} (%)	RC^{sec} (%)	Best solution
1	10	4	20	20	solving
2	10	4	40	40	breaking
3	10	4	60	60	breaking
4	10	4	80	80	only solving

Table 4: Database example

1) Input computation: for a given network and its flows, the first step must be to obtain the inputs, i.e., we need to run the routing algorithm without turn prohibition first and check the presence or absence of cyclic dependencies.

2) Identification of best solution in the database: then, we attempt to identify the best solution using the database. We consider that due to the continuity and monotonicity of the tipping point variations, we can identify the best solution if, for the input set $I = \{i_1, ..i_n\}$:

$\exists$ two sets of parameters $P^m = \{p_1^m, .., p_n^m\}$ and $P^M = \{p_1^M, .., p_n^M\}$ with the same best method and such as:

$$\forall j \in \{1, .., n\}, p_j^m \leqslant i_j \leqslant p_j^M$$

For example, using Table 4, with $I = \{N = 10, L = 4, RC^{prim} = RC^{sec} = 50\%\}$, we use $P^2 = \{N = 10, L = 4, RC^{prim} = RC^{sec} = 40\%\}$ and $P^3 = \{N = 10, L = 4, RC^{prim} = RC^{sec} = 60\%\}$ to deduce that the best solution for I is breaking the cycle dependencies.

Additionally, we can use conclusions from Section 4.5 to select the best solution. Following are a few examples using Table 4:

- $I = \{N = 10, L = 4, RC^{prim} = RC^{sec} = 10\%\}$, we know that the tipping point is over 20%. Hence, we can conclude that the best solution is solving;
- $I = \{N = 9, L = 4, RC^{prim} = RC^{sec} = 40\%\}$, we know that the tipping point decreases when N decreases and the only-solving area is not affected by N (see test numbers 2 and 4 in Fig. 10(c)). Hence, we can conclude that I is in the best=breaking area;
- $I = \{N = 12, L = 4, RC^{prim} = RC^{sec} = 40\%\}$, we know the tipping point increases with N. Hence, we are unable to directly conclude which is the best solution from the data of Table 4.

3) Updating the database: if the best method cannot be identified using the database, we need to identify it by also running the breaking method and comparing the bounds of the two methods (if breaking is possible). After identifying the best method based on the comparison of the results, we can add the new data to the database and select the best routing.

Finally, we can run a check to delete redundant entries. For example with Table 4 and $I = \{N = 10, L = 4, RC^{prim} = RC^{sec} = 22\%\}$, we identify with step 3 that both solutions have the same bounds, making this point a tipping point. Consequently, input sets 1 and 2 can be replaced by the new data, as illustrated in Table 5.

input set id	N	L	RC^{prim} (%)	RC^{sec} (%)	Best solution
3	10	4	60	60	breaking
4	10	4	80	80	only solving
5	10	4	22	22	tipping point

Table 5: Updated database

5.3 Discussion

The presentation of the tipping point method highlighted the fact that the main challenge of the method is to correctly build the database to avoid data loss and redundant data. In particular, the construction of the logic needed to deduce the best solution. However, while complex, all the needed information has been provided.

With this method, we are able to select the best solution for solving or breaking cyclic dependencies. When building the database, this method necessitates running the two routing algorithms and both the breaking and solving methods. Then, thanks to the database, we only have to run the turn-prohibition routing if we identify the breaking method as the best one. Hence, we can obtain the best bounds while limiting the amount of computation time.

This method is mainly useful when needing to assess a large number of input sets. For example, in [15, 30], heuristic searches are used to find TT offsets enforcing both RC and TT deadlines in networks mixing RC and TT frames. So, at each step of the search, RC delay bounds must be computed. Thus, in the case of cyclic dependencies, our proposal can largely reduce the search time.

6 CONCLUSION

In this paper, we have reviewed the existing methods for 1) breaking cyclic dependencies and computing bounds in feed-forward networks; 2) computing bounds in cyclic dependent networks, i.e., solving cyclic dependencies. For the breaking method, we have selected the turn-prohibition algorithm [27] associated to the Turnnet algorithm [14] to break the dependencies, and SEP/TFA++ [8, 16] to compute delay and backlog bounds. For the solving method, we have selected both the Time Stopping Method [12] and the Backlog-based Method [20], and used the minimum of the resulting bounds.

We have done a performance analysis of these two methods to compare them inside ring topologies for both AFDX and TTEthernet networks. In particular, our sensitivity analysis focused on the impact of the ring length, the path length, and different ring loads (RC and TT). Our results showed that depending on these variables, either breaking or solving can give the best delay and backlog bounds. We have also shown that breaking is not always possible due to routing constraints or port overloading. In particular, we showed that on the one hand, selecting the best solution can improve the delay and backlog bounds by over 70%. On the other hand, selecting the wrong method can increase the bounds by over 500%. Hence, it is important to identify the optimum solution for a given scenario.

Consequently, we have proposed the Tipping Point Method, to identify the points where both methods have the same bounds and select the best solution depending on input parameters. Thanks to these lower bounds, the delays and backlogs can validate lower deadline and memory constraints, resulting in the possibility of certifying networks with higher utilization rates. Initial results are encouraging, but before implementing industrial applications, the tipping method must be validated on industrial topologies.

REFERENCES

[1] Airlines Electronic Engineering Committee. Aircraft Data Network Part 7, Avionics Full Duplex Switched Ethernet (AFDX) Network, ARINC Specification 664. Aeronautical Radio, 2002.

[2] Ahmed Amari and Ahlem Mifdaoui. Worst-case timing analysis of ring networks with cyclic dependencies using network calculus. In *Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2017.

[3] Ahmed Amari, Ahlem Mifdaoui, Fabrice Frances, Jérôme Lacan, David Rambaud, and Loic Urbain. AeroRing: Avionics Full Duplex Ethernet Ring with High Availability and QoS Management. In *European Congress on Embedded Real Time Software and systems*, 2016.

[4] Anne Bouillard. Stability and performance guarantees in networks with cyclic dependencies. *arXiv preprint arXiv:1810.02623*, 2018.

[5] Anne Bouillard, Laurent Jouhet, and Eric Thierry. Service curves in Network Calculus: dos and don'ts. Research report, INRIA, 2009.

[6] Frédéric Boulanger, Dominique Marcadet, Martin Rayroley, Safouan Taha, and Benoit Valiron. A time synchronization protocol for A664-P7. In *DASC*. IEEE, 2018.

[7] Marc Boyer, Hugo Daigmorte, Nicolas Navet, and Jörn Migge. Performance impact of the interactions between time-triggered and rate-constrained transmissions in ttethernet. In *8th European Congress on Embedded Real Time Software and Systems*, pages 1–10, Toulouse, FR, 2016.

[8] Marc Boyer, Jörn Migge, and Nicolas Navet. An efficient and simple class of functions to model arrival curve of packetised flows. In *Proc. of the 1st Int. Workshop on Worst-Case Traversal Time (WCTT'2011)*, novembre 2011.

[9] Marc Boyer, Nicolas Navet, and Marc Fumey. Experimental assessment of timing verification techniques for afdx. In *6th European Congress on Embedded Real Time Software and Systems*, 2012.

[10] Henning Butz. The Airbus approach to open integrated modular avionics (IMA): technology, methods, processes and future road map. In *Workshop on Aircraft System Technologies (AST)*, 2007.

[11] Silviu S. Craciunas and Ramon Serna Oliver. Combined task- and network-level scheduling for distributed time-triggered systems. *Real-Time Systems*, 52(2):161–200, 2016.

[12] Rene L Cruz. A calculus of delay Part II: Network analysis. *IEEE Trans. Inform. Theory*, 37(1):132–141, 1991.

[13] Jonas Diemer, Daniel Thiele, and Rolf Ernst. Formal worst-case timing analysis of Ethernet topologies with strict-priority and AVB switching. In *Proc. International Symposium on Industrial Embedded Systems (SIES)*. IEEE Computer Society, 2012.

[14] Markus Fidler and Gerrit Einhoff. Routing in turn-prohibition based feed-forward networks. In *International Conference on Research in Networking*. Springer, 2004.

[15] Anaïs Finzi and Silviu S. Craciunas. Integration of SMT-based scheduling with RC network calculus analysis in TTEthernet networks. In *Proc. of International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019.

[16] Fabrice Frances, Christian Fraboul, and Jérôme Grieu. Using network calculus to optimize the AFDX network. In *Embeeded Real Time Software and Systems (ERTS)*, 2006.

[17] Jérôme Grieu. *Analyse et évaluation de techniques de commutation Ethernet pour l'interconnexion des systèmes avioniques*. PhD thesis, INPT, 2004.

[18] Institute of Electrical and Electronics Engineers, Inc. Time-Sensitive Networking Task Group. http://www.ieee802.org/1/pages/tsn.html, 2016. retrieved 06-Jul-2017.

[19] Jean-Yves Le Boudec. A theory of traffic regulators for deterministic networks with application to interleaved regulators. *IEEE/ACM Transactions on Networking (TON)*, 26(6):2721–2733, 2018.

[20] J.Y. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*, chapter 1. Springer-Verlag, 2001.

[21] Andres Mejia, Jose Flich, Jose Duato, S-A Reinemo, and Tor Skeie. Segment-based routing: An efficient fault-tolerant routing algorithm for meshes and tori. In *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2006.

[22] Ahlem Mifdaoui and Thierry Leydier. Beyond the Accuracy-Complexity Tradeoffs of Compositional Analyses using Network Calculus for Complex Networks. In *10th International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, 2017.

[23] José Carlos Sancho, Antonio Robles, and José Duato. A new methodology to compute deadlock-free routing tables for irregular networks. In *International Workshop on Communication, Architecture, and Applications for Network-Based Parallel Computing*. Springer, 2000.

[24] Jens B Schmitt and Frank A Zdarsky. The disco network calculator: a toolbox for worst case analysis. In *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*. ACM, 2006.

[25] Jens B Schmitt, Frank A Zdarsky, and Ivan Martinovic. Improving performance bounds in feed-forward networks by paying multiplexing only once. In *Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB)*. VDE, 2008.

[26] Michael D. Schroeder, Andrew D Birrell, Michael Burrows, Hal Murray, Roger M. Needham, Thomas L. Rodeheffer, Edwin H. Satterthwaite, and Charles P. Thacker. Autonet: A high-speed, self-configuring local area network using point-to-point links. *IEEE Journal on Selected Areas in Communications*, 9(8):1318–1335, 1991.

[27] David Starobinski, Mark Karpovsky, and Lev A Zakrevski. Application of network calculus to general topologies using turn-prohibition. *IEEE/ACM Transactions on Networking (TON)*, 11(3):411–421, 2003.

[28] Wilfried Steiner. An Evaluation of SMT-based Schedule Synthesis For Time-Triggered Multi-Hop Networks. In *RTSS*. IEEE Computer Society, 2010.

[29] Wilfried Steiner, Günther Bauer, Brendan Hall, Michael Paulitsch, and Srivatsan Varadarajan. TTEthernet Dataflow Concept. In *Eighth IEEE International Symposium on Network Computing and Applications*, 2009.

[30] Domitian Tamas-Selicean, Paul Pop, and Wilfried Steiner. Synthesis of communication schedules for TTEthernet-based mixed-criticality systems. In *Proc. CODES+ISSS*. ACM, 2012.

[31] Daniel Thiele, Philip Axer, and Rolf Ernst. Improving formal timing analysis of switched ethernet by exploiting fifo scheduling. In *Proceedings of the 52nd Annual Design Automation Conference*, page 41. ACM, 2015.

[32] Luxi Zhao, Paul Pop, Qiao Li, Junyan Chen, and Huagang Xiong. Timing analysis of rate-constrained traffic in TTEthernet using network calculus. *Real-Time Systems*, 53(2):254–287, 2017.

[33] Luxi Zhao, Huagangh Xiong, Zhong Zheng, and Qiao Li. Improving worst-case latency analysis for rate-constrained traffic in the time-triggered ethernet network. *IEEE Communications Letters*, 18(11):1927–1930, 2014.