Integration of SMT-based Scheduling with RC Network Calculus Analysis in TTEthernet Networks

Anaïs Finzi
TTTech Computertechnik AG
Schoenbrunner Strasse 7
A-1040 Vienna, Austria
anais.finzi@tttech.com

Silviu S. Craciunas
TTTech Computertechnik AG
Schoenbrunner Strasse 7
A-1040 Vienna, Austria
silviu.craciunas@tttech.com

***Abstract*—In mixed-criticality Ethernet-based time-triggered networks, like TTEthernet, time-triggered communication (TT) coexists with rate-constrained (RC) and best-effort (BE) traffic. A global communication scheme, i.e., a schedule, establishes contention-free transmission times for TT flows ensuring guaranteed low latency and minimal jitter. Current approaches use Satisfiability Modulo Theories (SMT) to formulate the scheduling constraints and solve the resulting problem. However, these approaches do not take into consideration the impact of the TT schedule on RC traffic. Hence, the resulting TT schedule may cause the worst-case latency requirements of RC traffic not to be fulfilled anymore.**

In this paper, we present a novel method for including an RC analysis in state-of-the-art SMT-based schedule synthesis algorithms via a feedback loop in order to maintain the optimality properties of the SMT-based approaches while also being able to improve the RC traffic delays. Our method is designed in such a way that it can be readily integrated into existing SMT- or MiP-based solutions. We evaluate our approach using variants derived from a realistic use-case and present methods to further improve the efficiency of our feedback-based approach.

I. Introduction

For certain application domains, critical communication flows need to be proven correct in terms of their temporal behavior. For example, in the aerospace domain, but also in emerging industrial automation systems, authorities require the proof of correctness as part of the certification process with respect to critical traffic fulfilling end-to-end latency requirements. These requirements have been guaranteed through analysis methods like Network Calculus [1], [2], [3] or the more recent Compositional Performance Analysis [4], for technologies like Avionics Full DupleX (AFDX) [5]. The Network Calculus method [1] is a well-known mathematical framework based on min-plus algebra that is widely used in the certification process to derive worst-case end-to-end latency bounds for individual asynchronous communication flows.

TTEthernet (SAE AS6802 [6], [7]) enhances the rate-constrained traffic class (RC) of AFDX with a fully synchronous time-triggered communication paradigm (TT) that offers better guarantees in terms of deterministic real-time temporal behavior as well as composability. For the TT traffic class, determinism is ensured via offline communication schedule that enforces a contention-free and precise delivery of critical frames across a switched multi-hop network within defined latency and jitter bounds.

The schedule generation for the TT traffic class is done either through heuristic-based approaches [8] or through optimal algorithms based on MiP or SMT [9], [10]. While the integration of RC analysis through network calculus is straightforward in heuristic-based approaches, there is no guarantee that a solution is found (or, if none exists, that a definitive negative result can be given) or that the produced schedule is, indeed, the optimal one (due to local minima). Optimal algorithms on the other hand, while being exponential in runtime complexity, do not suffer from these drawbacks. However, the network calculus analysis cannot be readily included in the SMT constraint formulation. Nevertheless, many applications require the optimality criteria of SMT-based schedulers while still being able to guarantee response times for RC traffic.

In this paper we provide a method for including RC analysis in state-of-the-art SMT-based schedule synthesis methods in order to maintain the optimality properties of the algorithm while still being able to offer delay results based on network calculus. Our method features a feedback loop which guides the SMT solver towards generating schedules for the TT traffic class that, besides adhering to the correctness constraints, also provide schedulability results for the RC traffic class. The main idea of the presented incremental algorithm is to use the SMT-solver to compute new offsets for TT frames such that the impact (delay) of TT frames, represented by the arrival curve of the TT traffic, on RC traffic is reduced. We make use of the optimization capabilities of modern SMT solvers to guide the placement of TT frames. Since the impact of optimization objectives on the runtime of the SMT solver can be significant, we also present an approximation method for reducing the number of SMT assertions needed in order to increase the runtime performance of the scheduler. We evaluate our solution via real-world use-cases and show the effectiveness of our approach.

We begin by briefly introducing TTEthernet, SMT-based scheduling, and the Network Calculus framework in Section II. In Section III we describe our proposed method, followed by an evaluation in Section IV. We survey related work in Section V and conclude the paper in Section VI.

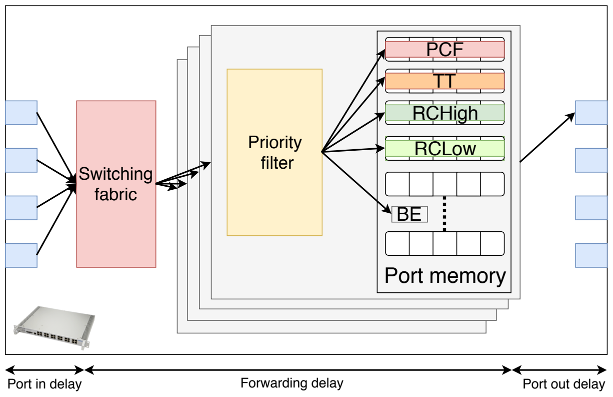

Fig. 1. Considered switch architecture

II. Theoretical Background

A. TTEthernet

TTEthernet is an extension to standard Ethernet currently used in mixed-criticality real-time applications in the aerospace domain but also in emerging industrial automation systems. TTEthernet features three types of traffic that represent different criticality levels: TT traffic (used for highly critical traffic with guaranteed end-to-end latency and minimal jitter), RC traffic with deterministic quality of service (QoS) guarantees and non-critical traffic (i.e. BE traffic). The TTEthernet [11] standard is based on the use of global time synchronization to send Time Triggered (TT) frames at precise, predefined times (encoded in a local schedule 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 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 remaining 4 lowest priorities are reserved for Best-Effort (BE) communication (cf. Fig. 1).

B. SMT-based TTEthernet schedule synthesis

Here, we briefly describe, based on [9], [10], [12] the SMT-based approach for computing communication schedules for TTEthernet.

Satisfiability Modulo Theories (SMT) is a well-known method, similar to SAT, used to check the satisfiability of first-order logical formulas based on a background theory such as linear integer arithmetic ($\mathcal{LA}(\mathbb{Z})$) or bit-vectors ($\mathcal{BV}$) [13], [14]. Current state-of-the-art TTEthernet scheduling approaches [10], use SMT solvers for generating the static schedule by generating assertions to the context of an SMT solver representing the necessary and sufficient correctness conditions. The result of the SMT solver is a solution for the given constraints, representing values for the offsets of individual frames on each device. Additionally, modern SMT solvers like Z3 [15] offer the possibility to include optimization criteria that find the optimal solution given minimization or maximization condition(s).

Our algorithm, which builds on top of the work in [9], uses SMT solvers to find out the schedule for a network based on the inherent TTEthernet technology constraints and optional user constraints. Steiner [9] proposes an incremental backtracking algorithm, which we also implemented, that splits the scheduling problem in small increments, scheduling subsets of flows at a time. If a partial solution is found for the subset, additional frames and constraints are added until either the complete schedule is found or a solution to a partial problem cannot be found. In the case of in-feasibility, the problem is backtracked and the size of the added subset is increased. In the worst case the algorithm backtracks to the root, scheduling the complete set of frames in one step. Our algorithm adds each flow from the input to the SMT context in sequence following the algorithm described in [9]. We refer the reader to [9] for a description and formalization of the correctness constraints for the SMT-based TTEthernet scheduler which we use in our method.

While the SMT-based approach retains the optimality property of SAT- or MiP-based solutions, the major downside is that the SMT solver acts as a black box and we are only able to influence it by defining constraints (assertions) and optimization criteria. Moreover, we are constrained to the expressiveness of first order logic. Hence, state-of-the-art SMT-based schedulers only take into account the TT flow constraints. As a result, the RC traffic can be strongly impacted by the placement of the scheduled TT frames and miss their required deadlines.

We propose a feedback-based approach that attempts to drive the SMT solver via optimization criteria towards placing TT frames on the timeline such that RC traffic will adhere to the latency and backlog requirements.

Please note that, while the proposed method is tailored to TTEthernet, the main feedback-based approach can also be used in Time-Sensitive Networks (TSN) which is the new standardized deterministic Ethernet solution for industrial communication systems. For TSN, existing SMT-based solutions (e.g. [12]) can be extended using our proposed method to consider AVB traffic shaped by CBS by including the network calculus-based AVB analysis results from [16].

C. Network Calculus

The timing analyses detailed in this paper are based on the Network Calculus framework [17] which 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 resource availability described by the so-called minimum *service curve* β, which represents the minimum amount of data that can be sent in any time interval.

Definition 1 (Arrival Curve): [17] 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:

$$\forall t, A(t) \leq (A \otimes {}^{1}\alpha)(t)$$

${}^{1}f \otimes g(t) = \inf_{0 \leq s \leq t}\{f(t-s) + g(s)\}$

Definition 2 (Strict minimum service curve): [17] The function β is the 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): [17] Consider a flow F constrained by an arrival curve α crossing a system $\mathcal{S}$ that offers a minimum service curve β. 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): [17] 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): [18] 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]:

$$(\beta - \sum_{j<i} \alpha_j - \max_{k \geq i} l_{k,max})_\uparrow$$

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 $\beta_{R,T}$, 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. The resulting output arrival curve is then: $\alpha^*(t) = r \cdot (t+T) + b$

An improvement of the leaky bucket shaper 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 [19]. 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 $\alpha(t) = min(C_{in} \cdot t, r \cdot t + b)$.

Extensions for the RC traffic class analysis for TTEthernet that consider the impact by the TT traffic class schedule have been introduced in [20], [21], [22].

III. Feedback Loop

An optimal solution would be to integrate the Network Calculus framework within the SMT solver to directly consider the constraints of the RC traffic. However, this is not possible due to the non-linearity of Network Calculus. In particular, the computation of the arrival curve of TT frames as described in [20] make use of several minimizations, maximizations and upper-bound values to compute the overall arrival curve of TT flows.

The current state-of-the-art SMT-based schedulers do not consider RC flows. However, the placement of the TT frames on the timeline may have a major impact on the delays experienced by RC flows since TT traffic has a higher priority. The main idea of our proposal (described in more detail below) is to use a feedback loop that uses the RC network calculus analysis to check the TT schedule and, if necessary, reschedule problematic flows.

First all the TT flows are scheduled using the optimization function of the SMT-solver to spread the TT frames. Our observation is that evenly spacing out TT frame placement leads to a lower impact on RC flows. Hence, we build an optimization metric that tries to evenly space out TT frames on the timeline and use optimization objectives to drive the scheduling and rescheduling of TT flows via the SMT solver.

Secondly, the RC analysis is called to determine if RC traffic fulfills the deadline requirements. If this is the case, the algorithm stops. If this is not the case, the algorithm identify the flow most likely to be causing delays and attempt to find a better offset, i.e., reschedules, using the optimization function of SMT.

This second step is repeated until an appropriate schedule is found or a stopping condition is reached.

The additional advantage of such a solution is that it is complementary to the existing implementation and can be readily integrated into existing tools. It does not require modification of the existing constraints, just the addition of optimization objectives and the implementation of the feedback loop.

A. General algorithm

The general algorithm is detailed in Alg. 1. To modify the values of the offsets, we propose to add an optimization constraint to the SMT scheduler, implemented in the function smtComputeOffset(flow) in Section III-C. In particular, we propose two methods for the computation of the optimization function. They are detailed in Sections III-D and III-E.

First, in Alg. 1, we set an initial offset for each flow (ranked from smallest to largest period) using smtCompute-Offset(flow), from Line 1 to Line 3. Then, we check whether the RC deadlines are fulfilled in Line 4. If it is not the case, we attempt to modify the offset of a flow selected by the function findBestFlow() (describe in Section III-B), in Lines 8 and 9.

If the new set of offsets has already been explored (Line 10), we use a while loop to explore possible offsets until all the flows have been tested or a new set has been found (Line 11). While no acceptable solution is found in Line 11, we add the flow to the diversification list, look for a new flow that is not in the diversification list, and compute the new offsets as before (Lines 12 to 15). If the new set of offsets has not already been explored, then the diversification list is reset (Lines 16 to 18). Finally, we check if the RC deadlines are now fulfilled and update the list of explored offsets (Lines 19 and 20).

[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] $g_\uparrow(t) = \max\{0, \sup_{0 \leq s \leq t} g(s)\}$

Algorithm 1: General algorithm

```
Require: $flows_{TT}$,$flows_{RC}$
1: for flow in $flows_{TT}$ do
2:     smtComputeOffset(flow)
3: end for
4: rcDeadlinesFulfilled=computeRCDelays($flows_{RC}$)
5: listExploredOffset=[]
6: diversification=[]
7: while rcDeadlinesFulfilled==False and
   diversification.size $<$ $flows_{TT}$.size do
8:     flow = findBestFlow()
9:     smtComputeOffset(flow)
10:    if getCurrentOffsets() in listExploredOffset then
11:        while getCurrentOffsets() in listExploredOffset and
           diversification.size $<$ $flows_{TT}$.size do
12:            diversification.add(flow)
13:            flow = findBestFlow(diversification)
14:            smtComputeOffset(flow)
15:        end while
16:    else
17:        diversification==[]
18:    end if
19:    rcDeadlinesFulfilled=computeRCDelays($flows_{RC}$)
20:    listExploredOffset.add(getCurrentOffsets())
21: end while
```

Algorithm 2: findBestFlow()

```
Require: $flows_{TT}$,$flows_{RC}$, $Data_{paths}$,diversification
Ensure: bestFlow
1: listCard={}
2: for flowTT in $flows_{TT}$ do
3:     listCard[flowTT]=0
4: end for
5: for flowRC in $flows_{RC}$ do
6:     for path in flowRC paths do
7:         for port in path do
8:             flowsTTInter=findIntersections()
9:             for flowTT in flowsTTInter do
10:                listCard[flowTT]+=1
11:            end for
12:        end for
13:    end for
14: end for
15: bestFlow=NULL
16: if not isEmpty(listCard) then
17:     bestFlow=findMaxCardinal(listCard,diversification)
18: end if
19: while bestFlow==NULL or bestFlow in diversification
    do
20:     bestFlow = $flows_{TT}$.getNext()
21: end while
```

B. *findBestFlow()*

The impact of TT flows is represented by the arrival curve of the TT traffic, which is computed using the formulas detailed in [20]. The main idea is to compute the impact of TT flows in all the possible situations and keep the maximum values, as illustrated in Fig. 2. The dotted lines are the different situations and the plain line staircase is the maximum of the dotted lines, representing the final TT flows maximum arrival curve, denoted $\alpha_{TT}^{k}(t)$.

From this staircase arrival curve, we are able to approximate a linear arrival curve. The rate of the linear curve is the same as the rate of the staircase curve over a period. Concerning the initial burst of the linear curve, it is computed such as the linear curve is always superior or equal to the staircase curve, with at least one intersection, as illustrated in Fig. 2. As a result, by modifying the value of the offset associated to the staircase intersection point, we may be able to reduce the value of the burst of the linear curve, and consequently, reduce the impact of TT traffic on RC flows. The intersection points are computed within the findIntersections() function.

To select the flow most likely to have the best impact (the algorithm is detailed in Algo. 2), we compute the number of times a flow is an intersection (from Line 5 to 14) and we select the flow with the most occurrences that is not in the diversification list, in Line 17. Finally, if no flow has been found, we select the first flow in the list of TT flows that is not in the diversification list, in Lines 19 to 21.

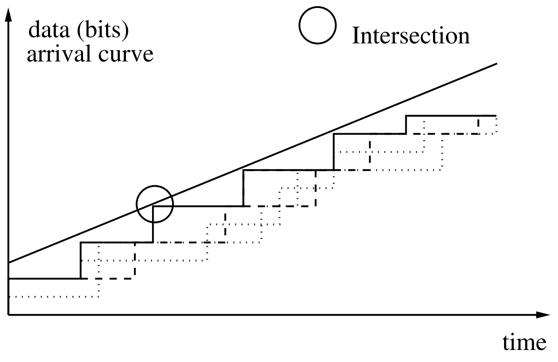

Fig. 2. Computing TT flows arrival curve $\alpha_{TT}^{k}(t)$

C. *smtComputeOffset()*

We consider a set of TT flows with a hyper-period HP (computed as the least common multiple of all flow periods). Our goal is to define the offset x of flow i with maximum frame sizes MFS_i and period T_i. All the preexisting constraints from [9] are enforced. In this section, we only present the constraints added for the optimization of the offsets. Additionally, if the flow has already been scheduled, we also add the constraint that x must differ from the current offset.

We denote C_{out} the capacity of the output port. For each TT frame k already scheduled, we define t_k^{start} and t_k^{end} the start and the end of the transmission. We have $m_k = \frac{t_k^{end}+t_{k+1}^{start}}{2}$. The function newSymbolicInt() creates a symbolic variable that will be used as an unknown by the SMT solver.

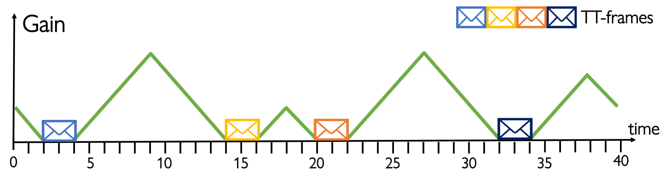

Fig. 3. Method 1: gain computation

In Alg. 3, for each port in each path of the TT flow i, we compute the constraints defining the gains, then they are added to the existing SMT constraints in Line 6. Finally offsets are computed by the SMT solver in Line 7. The computation of the new constraints in a port (i.e. Line 4) is described in Algo. 4.

Algorithm 3: Constraint computation for a path of TT flow i

```
Require: $TT\ flow\ i$, $path$
Ensure: $Offsets_i$
1: sumGain=newSymbolicInt()
2: constraints=preexisting constraints from [9]
3: for port in path do
4:     ($constraints$, $sumGain$)=computeConstraintInPort
       ($TTflow\ i$, $constraints$, $sumGain$, $Data_{port}$)
5: end for
6: constraints.add(maximize($sumGain$))
7: $Offsets_i$=smtSolve(constraints)
```

D. computeConstraintsInPort(): variant 1

To compute the offset leading to the minimum impact of TT flows on RC flows, we must spread the frames over the hyper-period in order to reduce the initial burst of the aggregated TT flows.

To achieve this goal, in each output port with TT flows, we define gain functions between the already scheduled TT frames, as illustrated in Fig. 3, with $T_i = 10$ and an Hyper-period $HP = 40$. We denote t_k^{end} the end of a frame transmission and t_{k+1}^{start} the start of the next transmission. For each TT frame $k \neq i$ already scheduled, the gain functions are determined by:

$$\forall t_k^{end} \leq t < \frac{t_k^{end}+t_{k+1}^{start}}{2}, gain(t) = t - t_k^{end}$$
$$\forall t_{k+1}^{start} \geq t \geq \frac{t_k^{end}+t_{k+1}^{start}}{2}, gain(t) = t_{k+1}^{start} - t$$

Alg. 4 describes the definition of the gain function illustrated in Fig. 3. In particular, Line 7 defines the outer bounds of a triangle, Line 8 (resp. Line 9) defines the right (resp. left) side of the triangle.

However, this method requires a large number of assertions since SMT optimization constraints can only be defined as linear curves in $\mathcal{LA}(\mathbb{Z})$. Hence, in between two frames, 6 assertions are necessary to define the two linear parts of the gain, i.e., to define the upper and lower bounds of x and the value associated to the gain. When the number of TT flows increases, so does the number of assertions needed, resulting in an increased runtime of the scheduler [10].

Consequently, we propose a second variant of the algorithm to reduce the number of assertions through preprocessing and hence improve the runtime of our approach.

Algorithm 4: Constraint computation in a port p: computeConstraintsInPort()

```
Require: $TT\ flow\ i$, $constraints$, $sumGain$, $Data_p$
Ensure: $constraints$, $sumGain$
1: x= newSymbolicInt()
2: $J = \frac{HP}{T_i}$
3: for j in range(0,J) do
4:     $gain_j$= newSymbolicInt()
5:     for frame $k$ in TT flows do
6:         if $t_k^{end} >= j \cdot T_i$ and $t_{k+1}^{start} <= (j+1) \cdot T_i$ then
7:             $A = $ And($x + j \cdot T_i + \frac{MFS_i}{C_{out}} <= t_k^{end}$ , $t_{k+1}^{start} <=$
               $x + j \cdot T_i$)
8:             $B = $ And($x + j \cdot T_i >= m_k$ , $gain_j =$
               $t_{k+1}^{start} - x - j \cdot T_i$)
9:             $C = $ And($x + j \cdot T_i <= m_k$ , $gain_j =$
               $x + j \cdot T_i - t_k^{end}$)
10:            $constraints.add$(And($A$, Or($B$, $C$)))
11:            $sumGain+ = gain_j$
12:        end if
13:    end for
14: end for
```

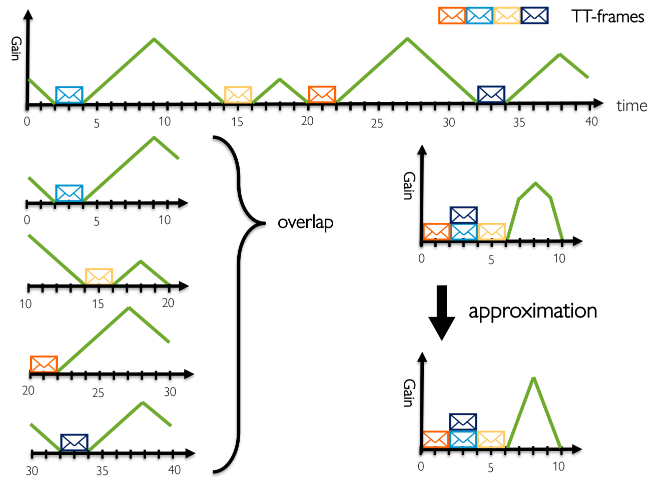

Fig. 4. Method 2: gain computation

E. computeConstraintsInPort(): variant 2

The main idea of the improvement is to consider only the period T_i of the flow i and compute the values excluded by the currently scheduled frames, i.e., if a frame is being transmitted, or if the inter-frame gap is too small to transmit the frame of flow i. Then, the gains (only on the acceptable times) are summed up, as illustrated in Fig. 4. Finally, to further reduce the number of linear parts of the gain function (which means reducing the number of assertions and hence the computation time), we approximate the remaining gain functions to keep only a maximum of two linear functions per interval, as illustrated in Fig. 4, with $T_i = 10$ and $HP = 40$.

In the example described in Fig. 3 and Fig. 4, the number of assertions has been reduced from 24 to 6, which should improve the timing performances. In the next section, we will present experimental results to validate these concepts and assess the gain in performance between the two variants.

IV. Performance analysis

In this section, we do a performance analysis based on a real-world use-case in order to compare the two selected variants of our solution. First, we present preliminary assumptions, e.g., the model of the flow, the TTEthernet switch and end-systems, and delay computations. Then, after presenting our case study, we do a comparison of the two methods for a TTEthernet network.

A. Preliminaries and assumptions

Switch and End-System model: we consider the TTEthernet switch architecture described in Fig. 1. The input port delay is the amount of time necessary for a frame i to 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 $n \in \{sw, es\}$. All the flows are using the shuffling integration policy in the switches and end-systems, i.e, the TT frames can be delayed by lower priority frames.

TTEthernet output ports: the impact of the TT traffic on RC traffic is computed using the input arrival curves proposed in [20]. Then, the service curves are computed using Th. 3.

Traffic model: to compute the delay bounds within each node (output port, switch sw or end-system es), we use Th. 1 under the following assumptions: (i) staircase 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). The initial arrival curve sent by the traffic source is $\alpha_I(t) = \sum_{i \in I} MFS_i \cdot \lceil \frac{t+J_i}{BAG_i} \rceil$. For each class I, the aggregate traffic has an input arrival curve in the node $n \in \{es, sw\}$: $\alpha^n_I(t) = \min\{C^n_{I,in} \cdot t, \sum_{i \in I, i \in n} \alpha^n_i(t)\}$, 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, and $\alpha^n_i(t)$ is the input arrival curve of flow i in node n. $\alpha^n_i(t)$ is a staircase curve as illustrated in Fig. 5, with WCD_{prec} the delays in the previous nodes. We considered that in the generating node n, $C^n_{I,in} = +\infty$; (ii) the offered service curve by node n to the traffic class I is a staircase curve as illustrated in Fig. 5. This results from using Th. 3 with staircase arrival curves.

Delay bound computation: 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:

$$\alpha^{port}_I(t) = \min\{C^n_{I,in} \cdot (t + \delta^n_{fwd}), \alpha^n_I(t + \delta^n_{fwd})\}$$

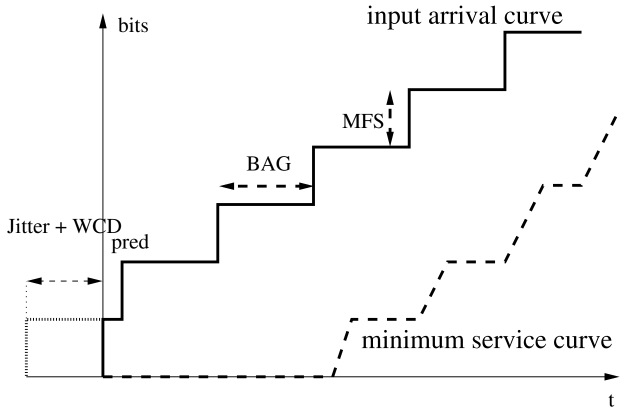

Fig. 5. Traffic model

with $\delta^n_{fwd} = WCD^n_{fwd} - BCD^n_{fwd}$. 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 switch (or end-system) n is then: $WCD^n_I = WCD^{port}_I + WCD^n_{fwd}$ and the output arrival curve is $\alpha^{n,*}_I(t) = \alpha^n_I(t + \delta^n_{fwd} + WCD^{port}_I)$.

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

B. Case Study

We consider a real-world project AERO1 [7] depicted in Fig. 6, with 3 end-systems, i.e., PG1, PG2 and PX2, and 3 switches, i.e., SW0, SW1, SW2. The forwarding delays are described in Table I, and the link capacity is 100 Mbps.

Node n	WCD^n_{fwd} (s)	BCD^n_{fwd} (s)
End System	$2.43 \cdot 10^{-6}$	$2.18 \cdot 10^{-6}$
Switch	$2.50 \cdot 10^{-6}$	$2.41 \cdot 10^{-6}$

TABLE I
Forwarding delays

We consider 12 flows for TT (resp. RC) traffic, with a redundancy level 3, i.e. 36 VLs, with identical period (resp. BAG) of 4 ms and MFS=1518 bytes. The RC traffic has a default maximum initial jitter of 100 μs. Six VLs are generated in PG1 with PG2 and PX2 as destinations. The other six are generated in PG2 with PG1 and PX2 as destinations. The default deadline of a flow is equal to the period.

In order to assess both methods, we explore different scenarios by varying two parameters, i.e., the TT deadline and RC jitter, as illustrated in Table II. As highlighted in the traffic model, increasing the initial jitter increases the input arrival curve of the generated traffic, resulting in the increase of the RC end-to-end delays for identical TT offsets. Consequently, increasing the RC jitter also increases the constraints on the RC deadline and delay. We choose to vary the jitter because

[7]Please note that the realistic test cases have been anonymized due to contractual obligations

it is a minor parameter compared to MFS and BAG, allowing us to remain close to the default scenario.

The results of various scenarios will highlight the advantages and drawbacks of each method.

We focus on the maximum number of iterations n_{max}, on the maximum time of execution t_{max}, on the iteration where the best result is found n_{best}, on the time necessary to find the best result t_{best}, and on the schedulability $sched$. We define the schedulability as the percentage of schedulable flows. Additionally, to ensure the algorithm ends within an acceptable time frame, we have added a constraint in Line 7 of Algo. 1 to limit the number of iterations to 2 000.

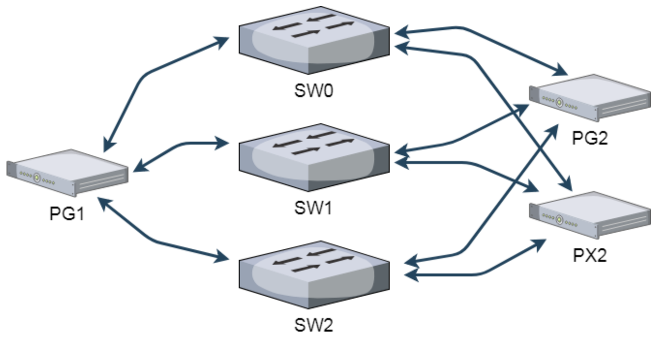

Fig. 6. AERO1 use-case

scenario	TT deadline (s)	RC Jitter (s)
default	0.004	$100{\cdot}10^{-6}$
1.1	0.004	$800{\cdot}10^{-6}$
1.2	0.004	$900{\cdot}10^{-6}$
1.3	0.004	$1000{\cdot}10^{-6}$
1.4	0.004	$1100{\cdot}10^{-6}$
1.5	0.004	$1200{\cdot}10^{-6}$
2.1	0.002	$100{\cdot}10^{-6}$
2.2	0.002	$300{\cdot}10^{-6}$
2.3	0.002	$400{\cdot}10^{-6}$

TABLE II
SCENARIOS DESCRIPTION

C. Results

First, we compute the results of all 9 scenarios without the optimization methods, i.e., with the current implementation. Results show that the computation is done in about t_{max}=1.80 s and that 18 out of 36 RC VLs are not schedulable, i.e., their delay is greater than their deadline. Hence, the schedulability of RC flows is 50% without the optimization methods for all 9 scenarios.

In Table III, we present the results of the scenarios finding an acceptable solution, i.e., all the RC flows are schedulable.

In Table IV, we present the results of scenarios not finding an acceptable solution, i.e., not all the RC flows are schedulable.

First, concerning the default scenario, we can see from Table III that both methods find solutions after the first iteration ($n_{max} = 1$). Hence, the schedulability is increased from 50%

scenario	method	n_{max}	t_{max} (s)
default	1	1	**101**
default	2	1	**2.5**
1.1	1	1	**95**
1.1	2	3	**3**
1.2	1	4	**174**
1.2	2	691	**256**
1.3	1	4	**170**
1.3	2	1110	**508**
2.1	1	8	**46**
2.1	2	1	**2.3**
2.2	1	1841	**3934**
2.2	2	123	**76**

TABLE III
RESULTS WITH A SCHEDULABILITY OF 100% FOR BOTH METHODS

scenario	method	n_{max}	t_{max} (s)	n_{best}	t_{best} (s)	sched
1.4	1	55	604	55	604	**100%**
1.4	2	2000	1190	1053	616	**83%**
1.5	1	2000	8621	1	115	**50%**
1.5	2	2000	1180	1535	882	**66%**
2.3	1	2000	**3968**	255	**891**	83%
2.3	2	2000	**1023**	660	**350**	83%

TABLE IV
RESULTS WITHOUT A SCHEDULABILITY OF 100% FOR BOTH METHODS

to 100%. Additionally, t_{max} shows that, as expected, method 2 is faster than method 1 (up to 51 times faster in scenario 2.2), and slightly slower than the current method without the optimization.

When studying all 9 scenarios, we notice that method 1 tends to find the best solutions in less iterations (i.e. n_{max} and n_{best} in scenarios 1.1, 1.2, 1.3, 1.4, and 2.3), but not always (i.e. scenarios 2.1 and 2.2). This is most likely due to the approximations done on the gain functions in method 2.

However, less iterations does not necessary result in better timing performances (i.e. t_{max} and t_{best} in scenarios 1.1 and 2.3) because, as expected, an iteration with method 2 is faster than with method 1, e.g. about 4 times faster in scenario 2.3 and up to 40 times faster in the default scenario.

When the schedulability is not 100% with both methods (see Table IV), we can see that the best method, i.e. with the best schedulability, depends on the scenario: method 1 for scenario 1.4 and method 2 for scenario 1.5. This again highlights that either method can find the best offsets.

Also, it is interesting to compare the maximum time to explore all 2 000 options: in scenario 1.5, it takes $t_{max} = 8621s$ for the first method to finish, and only 1180s for method 2 to assess that no solution gives 100% schedulability. Additionally, the schedulability with method 1, i.e., the longest to finish, is lower than with method 2, i.e., the fastest method.

Finally, we can conclude that while method 1 is less complex to implement and generally finds results in less iterations, method 2 is the best method to find solutions in an acceptable time frame. Additionally, we see that our feedback-based method is able to generate schedules for TT flows such that the schedulability of RC traffic is increased for all 9 scenarios (even up to 100%).

V. Related Work

The synthesis of time-triggered schedules has been initially introduced for the static cyclic task scheduling problem in [23]. Creating time-triggered schedules for TTEthernet networks by using SMT solvers was first proposed by Steiner in [9] and extended in [24], [10] to also generate the related cyclic schedule tables for tasks on end-system nodes. We use the constraints defined in [9] for our SMT-based scheduler implementation. An MiP-based approach is presented in [25] which generates task and messages schedules in switched time-triggered networks with multi-objective optimization.

Employing the network calculus framework for checking the worst-case latency of RC(AFDX) flows in TTEthernet networks has been addressed in [2], [20]. Specifically, we use the results in [20] for the analysis step of our feedback loop.

The work of Steiner [26] attempts to minimize the impact of TT frames on RC traffic via the concept of *schedule porosity* which leaves gaps in the TT schedule in order to minimize the starvation impact on RC frames. This method, as opposed to ours, does not make use of the network calculus framework to derive upper bounds on the worst-case latency of RC flows. The same RC analysis method as in [26] is used in [8] to drive a Tabu Search-based heuristic towards finding a schedule for TT frames while minimizing the end-to-end delay of rate-constrained frames. This method does not have the optimality criteria of SMT-based approaches since it relies on heuristics and, furthermore, does not use network calculus for the worst-case RC delay analysis.

In terms of analyzing the impact of time-triggered messages on RC traffic, several methods have been proposed. Including the TT traffic schedule in the RC service curve definitions of the network calculus framework has been presented in [20]. Boyer et al. [22] present an evaluation study of different traffic integration policies and the resulting TT impact on RC communication. Abuteir and Obermaisser [27] present a heuristic approach for allocating and scheduling mixed-criticality communication as well as a simulation and verification framework for TTEthernet networks. In [28], the authors evaluate the impact of different placement strategies for TT windows on the end-to-end delay and jitter of RC messages on the same path segment and validate the claims using simulations.

VI. Conclusion

We have presented a novel method for including RC analysis in state-of-the-art SMT-based TTEthernet scheduling algorithms that enforces RC traffic deadlines while maintaining the optimality properties of SMT-based approaches. Our method aims to drive the placement of the TT frames via optimization criteria given to the SMT solver in such a way that the impact of TT frames on RC traffic is reduced and hence, RC deadlines can be fulfilled. We evaluated our approach using variants derived from a realistic use-case and presented methods to further improve the efficiency of our feedback-based approach. In particular, results show that the schedulability of the RC traffic can be doubled, from 50% to 100%.

References

[1] J. Grieu, "Analyse et évaluation de techniques de commutation Ethernet pour l'interconnexion des systèmes avioniques," Ph.D. dissertation, INPT, 2004.

[2] F. Frances, C. Fraboul, and J. Grieu, "Using network calculus to optimize the AFDX network," in *Proc. ERTS*, 2006.

[3] J. Diemer, D. Thiele, and R. Ernst, "Formal worst-case timing analysis of Ethernet topologies with strict-priority and AVB switching," in *Proc. SIES*. IEEE Computer Society, 2012.

[4] D. Thiele, P. Axer, and R. Ernst, "Improving formal timing analysis of switched Ethernet by exploiting FIFO scheduling," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 41.

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

[6] SAE International, "SAE AS6802 Time-Triggered Ethernet," http://standards.sae.org/as6802/, 2011.

[7] W. Steiner, G. Bauer, B. Hall, and M. Paulitsch, "TTEthernet: Time-Triggered Ethernet," in *Time-Triggered Communication*, R. Obermaisser, Ed. CRC Press, Aug 2011.

[8] D. Tamas-Selicean, P. Pop, and W. Steiner, "Synthesis of communication schedules for TTEthernet-based mixed-criticality systems," in *Proc. CODES+ISSS*. ACM, 2012.

[9] W. Steiner, "An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks," in *Proc. RTSS*. IEEE, 2010.

[10] S. S. Craciunas and R. Serna Oliver, "Combined task- and network-level scheduling for distributed time-triggered systems," *Real-Time Systems*, vol. 52, no. 2, 2016.

[11] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, "The Time-Triggered Ethernet (TTE) Design," *Proc. ISORC*, 2005.

[12] R. Serna Oliver, S. S. Craciunas, and W. Steiner, "IEEE 802.1Qbv Gate Control List Synthesis using Array Theory Encoding," in *Proc. RTAS*. IEEE, 2018.

[13] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Satisfiability*. IOS Press, 2009, vol. 185.

[14] R. Sebastiani, "Lazy satisfiability modulo theories," *JSAT*, vol. 3, no. 3-4, pp. 141–224, 2007.

[15] N. Bjørner, A. Phan, and L. Fleckenstein, "νz - an optimizing SMT solver," in *Proc. TACAS*. Springer, 2015.

[16] L. Zhao, P. Pop, Z. Zheng, and Q. Li, "Timing analysis of AVB traffic in TSN networks using network calculus," in *Proc. RTAS*, 2018.

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

[18] A. Bouillard, L. Jouhet, and E. Thierry, "Service curves in Network Calculus: dos and don'ts," INRIA, Research Report, 2009.

[19] M. Boyer, J. Migge, and N. Navet, "An efficient and simple class of functions to model arrival curve of packetised flows," in *Proc. WCTT*, 2011.

[20] L. Zhao, P. Pop, Q. Li, J. Chen, and H. Xiong, "Timing analysis of rate-constrained traffic in TTEthernet using network calculus," *Real-Time Systems*, vol. 53, no. 2, pp. 254–287, 2017.

[21] L. X. Zhao, H. G. Xiong, Z. Zheng, and Q. Li, "Improving worst-case latency analysis for rate-constrained traffic in the time-triggered ethernet network," *IEEE Communications Letters*, vol. 18, no. 11, 2014.

[22] M. Boyer, H. Daigmorte, N. Navet, and J. Migge, "Performance impact of the interactions between time-triggered and rate-constrained transmissions in TTEthernet," in *Proc. ERTS*, 2016.

[23] T. P. Baker and A. Shaw, "The cyclic executive model and Ada," *Real-Time Systems*, vol. 1, no. 1, pp. 7–25, 1989.

[24] S. S. Craciunas and R. Serna Oliver, "SMT-based task- and network-level static schedule generation for time-triggered networked systems," in *Proc. RTNS*. ACM, 2014.

[25] L. Zhang, D. Goswami, R. Schneider, and S. Chakraborty, "Task- and network-level schedule co-synthesis of Ethernet-based time-triggered systems," in *Proc. ASP-DAC*. IEEE Computer Society, 2014.

[26] W. Steiner, "Synthesis of static communication schedules for mixed-criticality systems," in *Proc. ISORCW*. IEEE Computer Society, 2011.

[27] M. Abuteir and R. Obermaisser, "Scheduling of rate-constrained and time-triggered traffic in multi-cluster TTEthernet systems," in *Proc. INDIN*, 2015.

[28] F. Heilmann and G. Fohler, "Impact of time-triggered transmission window placement on rate-constrained traffic in TTEthernet networks," *SIGBED Rev.*, vol. 15, no. 3, pp. 7–12, Aug. 2018.