A Hybrid Approach to WCTT Analysis in a Real-Time Switched Ethernet Network

Aakash Soni, Jean-Luc Scharbarg, Jérôme Ermont

A Hybrid Approach to WCTT Analysis in a Real-Time Switched Ethernet Network

Aakash Soni
LyRIDS, ECE Research Center
Paris, France
aakash.soni@ece.fr

Jean-Luc Scharbarg
IRIT-ENSEEIHT
Toulouse, France
jean-luc.scharbarg@enseeiht.fr

Jérôme Ermont
IRIT-ENSEEIHT
Toulouse, France
jerome.ermont@enseeiht.fr

***Abstract*—Real-time Ethernet has become a popular solution for handling critical communication in embedded systems, partly thanks to the availability of safe worst-case traversal time analysis. These analyses scale well, but provide pessimistic upper bounds on end-to-end delays leading to over-dimensioning of the network architecture. Computing worst-case delays requires an exhaustive search over all possible scenarios that quickly leads to a combinatorial explosion. Consequently, it is limited to network configuration with less than 100 flows, far from industrial-size ones with more than 1000 flows. In this paper, we propose a hybrid approach (HA) that (1) reduces the number of scenarios to be analysed to get the worst-case delay, (2) leverages exact delays in order to tighten bounds on worst-case delays when the reduced number of scenarios is still too big to be completely analysed. On a realistic avionics case study, HA proved scalable for worst-case delay computations and reduced upper-bound pessimism by over 40%.**

***Index Terms*—WCTT analysis, Network Calculus, Model Checking, Hybrid approach, Switched Ethernet.**

I. Introduction

Real-time Ethernet has become a popular networking solution to cope with the increasing demand of critical data transmissions with hard real-time constraints in the context of embedded systems. The computation of an upper limit is mandatory to ensure that no deadline will be missed. Such a Worst-Case Traversal Time (WCTT) analysis is a required part of the certification process of safety critical systems, such as an avionic data network based on Avionic Full DupleX switched Ethernet (AFDX). In this paper we focus on AFDX.

The traversal time of a frame transmitted on the network highly depends on the waiting delays in output buffers, thus on the instantaneous pending traffic in the corresponding output port. Since the AFDX network is globally asynchronous (no common clock for switches and end-systems), the arrival of frames from different flows in an output port cannot be predicted in advance, and every possible combination of frame arrivals must be checked to determine the worst-case delay.

Some work has been devoted to the computation of this worst-case delay [1]–[5]. It is based on Model Checking (MC) and timed automata [6]. It implements an exhaustive search on all candidate scenarios for the worst-case delays. Additionally [7] derives and implements an algorithm which explores all these candidate scenarios. This algorithm allows the computation of worst-case delays for network configurations with up to 60 flows.

For larger configurations the computation does not finish, since the number of candidate scenarios is intractable. However this computation can be stopped after a predefined duration so that only a part of candidate scenarios are analysed. Therefore, it provides an under estimation of the worst-case delay. Nevertheless, up to now no approach can compute exact worst-case delays for industrial-size configurations.

Alternative approaches, such as Network Calculus (NC) [8], Trajectories [9], [10], Forward Analysis (FA) [11] and Compositional Performance Analysis (CPA) [12] are based on conservative models. They compute upper bounds on the worst-case end-to-end delays based on overestimation of the traffic and/or underestimation of the service offered by network components. On the positive side, these approaches scale well: they are able to analyse large scale systems with thousands of data flows exchanged over hundreds of nodes. On the negative side, they compute pessimistic upper bounds, leading to over-dimensioning of the network. Nevertheless, NC has been successfully used to dimension and certify the AFDX network of Airbus A380 and A350 aircraft. Moreover, some advances have been made in the context of AFDX network to reduce the pessimism [10], [13]. In [10] improvements in delay bounds are proposed by considering the serialisation effect, i.e. frames coming from the same input link cannot arrive in a switch at the same time. [13] integrates the effect of temporal separation of flows at source end-nodes which result in significant improvement in delay bound computation.

The remaining pessimism in the NC approach is analysed in [14] for industrial AFDX networks. On average, it can be as high as 11.5%. This remaining pessimism is a matter of concern since it contributes to increased cost and inefficient utilisation of the network.

In this paper we propose the Hybrid Approach (HA). It proceeds iteratively. First it reduces the number of scenarios to be considered for worst-case analysis. Second it leverages exact delay analysis to tighten upper bounds when this number of scenarios remains too high.

The contributions of this paper are the following:

- design and implementation of the Hybrid Approach which takes advantage of both exact delay and upper bound computations in order to provide exact worst-case delay or tight upper bounds,

- evaluation of the approach on a realistic industrial-size network configuration.

The rest of the paper is organised as follows. Section II summarises the context of the study, i.e. the AFDX network and its state-of-the-art timing analysis, and states the problem. The Hybrid Approach proposed in this paper is presented in Section III and evaluated in Section IV. Section V concludes the paper and gives some direction for future work.

II. Context

A. AFDX Network

An AFDX network is composed of a set of end-systems interconnected by switches via full-duplex links. Each end-system is connected to exactly one switch port. The typical link bandwidth is R = 100 Mbps. Each data flow transmitted on the network is statically defined as a virtual link (VL) v_i with the following features:

- its Bandwidth Allocation Gap (BAG_i), i.e. the minimum duration between two consecutive frames at its source end-system,
- its maximum (l_i^{max}) and minimum (l_i^{min}) frame length,
- its statically defined unidirectional multicast path $\mathcal{P}_i$.

The forwarding process in each switch is based on a static table. It introduces an upper bounded latency sl. Frames are served through First-In-First-Out (FIFO) queuing.

Figure 1 shows a small illustrative AFDX network. It is composed of six end-systems ($e_1 - e_6$) interconnected by two switches (S_1, S_2). Ten VLs ($v_0 - v_9$) are transmitted on this network. Their features are summarised in Table I.

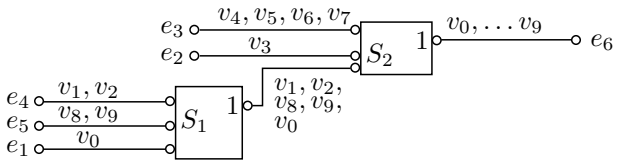


Fig. 1. Switched Ethernet Network Example

TABLE I
VL parameters in network example (Figure 1)

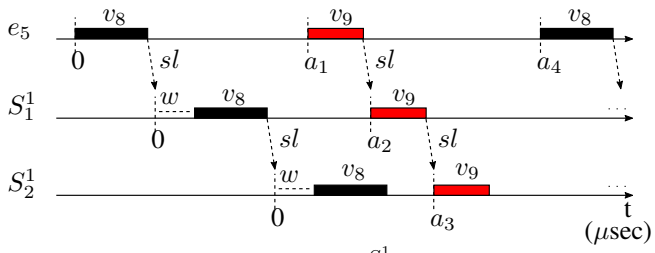
VL	BAG (μsec)	l_i^{max} ($bytes$)	tr_i (μsec)	$O_{d,i}^{e_x}$ (μsec)
v_0	128000	107	8.56	0
v_1	32000	171	13.68	8000
v_2	16000	307	24.56	0
v_3	64000	155	12.40	0
v_4	32000	543	43.44	24000
v_5	128000	263	21.04	4000
v_6	32000	571	45.68	8000
v_7	16000	407	32.56	0
v_8	32000	343	27.44	0
v_9	128000	263	21.04	16000

AFDX network does not implement a global synchronization. Therefore, clocks of different end-systems are fully independent and the offset between two VLs generated by two different end-systems is unknown. It depends on the starting instant of each end-system and the drift between their clocks. Hereby, no assumption can be made on this offset.

Conversely, VLs generated by a given end-system are scheduled, based on the clock of this end-system. This local scheduling has to ensure a bounded jitter (at most 500 μs) for each VL at source end-system [15]. It guarantees that the minimum duration between two consecutive transmissions of a given VL by its source end-system is never less than its BAG minus 500 μs. Thus, the first switch on the VL path is able to check that the traffic contract of the VL is not violated.

This jitter constraint can be ensured by Time Division Multiple Access (TDMA) scheduling [16]. Periodic slots are allocated to each VL, fully eliminating the jitter. Such a TDMA scheduling can be modelled by the release time of the first frame of each VL (corresponding to its first allocated slot), called its definite offset [17]. It is noted $O_{d,i}^{e_x}$ for VL v_i generated by end-system e_x. Definite offsets for the configuration in Figure 1 are given in Table I.

Due to this scheduling, there is a temporal separation between VLs. It is illustrated in Figure 2 for end-system e_5 of the configuration in Figure 1. e_5 is the source of VLs v_8 and v_9. From the last column in Table I, definite offsets of v_8 and v_9 are $O_{d,8}^{e_5} = 0$ μ s and $O_{d,9}^{e_5} = 16000$ μs. Therefore, the first frames of v_8 and v_9 start transmission at $t = 0$ μs and $t = 16000$ μs. Since $BAG_8 = 32000$ μs, the second frame of v_8 starts transmission at $t = 32000$ μs. Then one v_8 frame is transmitted every 32000 μs and the whole sequence is repeated after 128000 μs (hyperperiod of v_8 and v_9). Hereby, the temporal separation between one frame of v_8 and frame of v_9 is never less than 16000 μs. This minimum temporal separation from v_8 to v_9 at node e_5 is called relative offset and noted $O_{r,8,9}^{e_5}$. The three first transmissions are depicted in Figure 2 (top).



$a_1 = O_{d,9}^{e_5} = 16000$, $a_2 = O_{r,8,9}^{S_1^1} = 15993.6$, $a_3 = O_{r,8,9}^{S_2^1} = 15954.1$, $a_4 = BAG_8 = 32000$, w = waiting time in buffer, sl = switching latency.

Fig. 2. Temporal separation of e_5 VLs with benchmark v_8

The temporal separation between VLs is propagated as long as the VLs follow the same path. However, it can be reduced, since different frames might experience different delays. This phenomenon is illustrated in the arbitrary scenario depicted in Figure 2. At the switch output port S_1^1, v_8 is delayed by VLs from e_4 and e_1, due to FIFO scheduling, while v_9 is not.

Similarly, at the output port S_2^1, v_8 is delayed by VLs from e_3 and e_2, while v_9 is not. Therefore, the temporal separation between the first frames of v_8 and v_9 is reduced.

Since VLs transmit critical data, their maximum end-to-end delay has to be determined. To that purpose, different methods for the Worst-Case Traversal Time analysis of VLs have been proposed. A first set of works has been devoted to the computation of the exact worst-case end-to-end delay of a VL. It is presented in the following section.

B. Exact WCTT analysis for AFDX

The end-to-end delay experienced by a frame transmitted on an AFDX network is composed of the following parts:

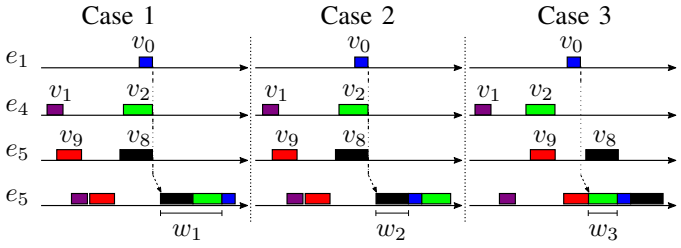
- the transmission time (tr) on the links, which is directly computed from the frame size and the link bandwidth,
- the switching latencies in each crossed switch, which is upper-bounded by sl per switch,
- the waiting delays in the queues of the crossed output ports, which depends on the pending frames in these queues at the arrival time of the frame under study.

Considering the configuration in Figure 1, Figure 3 shows 3 possible scenarios of waiting delay for one v_0 frame at S_1^1. Due to FIFO scheduling, a frame must wait in the queue until the frames before it are served.

In case 1 in Figure 3, v_0 is queued after v_2 and v_8, and thus it experiences a waiting delay $w_1 = tr_{v_8} + tr_{v_2}$ = 52 μsec.

Case 2 is similar to case 1 except that v_0 is queued before v_2. Hereby, the waiting delay for v_0 in the queue is $w_2 = tr_{v_8}$ = 27.44 μsec.

In case 3, v_2 is generated earlier while v_9 and v_8 are generated later. v_8 doesn't delay v_0 any more since it arrives after v_0 at S_1^1. Conversely, on the arrival of v_0 there is a frame from v_2 still waiting to be served. So the waiting delay for v_0 in this case is $w_3 = tr_{v_2}$ = 24.56 μsec.



w_1 = 52 μsec, w_2 = 27.44 μsec, w_3 = 24.56 μsec.

Fig. 3. Few of the possible scenarios of waiting delay for v_0 at S_1^1

These scenarios clearly illustrate that the delay experienced by a given VL (in this case v_0) depends upon the relative generation time of frames sharing a path in the network and also upon their order of queuing at the shared output port.

Therefore, the WCTT analysis has to identify one scenario that maximizes the overall waiting delay in the queues crossed by the frame under study. [7] proves that a scenario cannot be candidate to the worst-case if it doesn't verify the *critical instant* property: the frame under study arrives in each output port on its path at the same instant as one frame from every other input link of the corresponding switch and the frame under study is queued last. The competing VLs, whose frames participate in delaying the frame under study at the critical instant, are called benchmarks. Coming back to Figure 3, case 2 cannot be candidate to the worst-case, since v_0 is queued before v_2. Neither can case 3, since no frame from e_4 arrives at the same time as v_0. Conversely, case 1 is a worst-case candidate.

Based on the critical instant property, let us observe the worst-case candidate scenarios for v_0 on the path e_1–S_1^1–S_2^1–e_6. For the sake of simplicity, assume $sl = 0$. At the source end-system e_1, v_0 is the only VL. Otherwise, at the source end-system, VLs are separated by offsets and cannot delay each other, which always leads to only one scenario. At S_1^1, since v_0 competes with two VLs from e_4 and two VLs from e_5, there are four candidate scenarios at the switch port S_1^1. The frame sequences obtained at S_1^1 in these scenarios are depicted in Figure 4. Additionally, v_0 competes in S_2 with 4

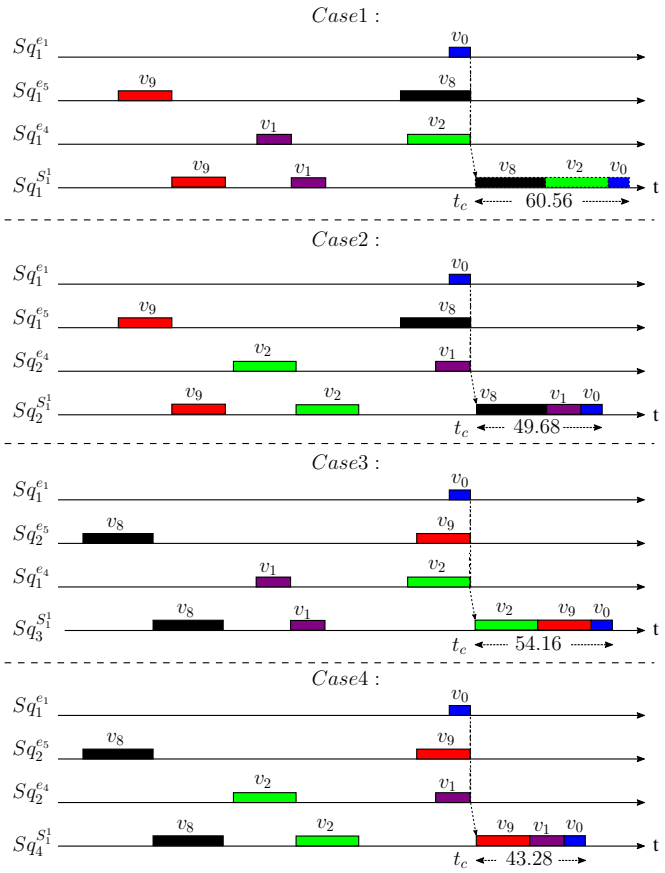


Fig. 4. Frame sequence combinations for v_0 at S_1^1

VLs from e_3 and 1 VL from e_2, leading to four candidate scenarios at switch port S_2^1. This means, overall we have 16 candidate scenarios : 4 scenarios at S_2^1 for each frame sequence obtained in the 4 scenarios at S_1^1. These scenarios are depicted by an inverted tree in Figure 5.

Each terminal node (leaf) of this tree corresponds to a can-

didate scenario which is identified by a label. For instance, the left-most leaf is labelled $v8, v1, v4, v3$: in the corresponding scenario one v_0 frame arrives at the same instant as frames from benchmark VLs v_8 and v_1 in the S_1^1 output port and frames from benchmark VLs v_4 and v_3 in the S_2^1 output port. Each internal node (branch point of the tree) aggregates candidate scenarios in its subtree. For example, the node with label $v8, v2, *, *$ models the four cases where $v0$ frame arrives at the same instant as frames from $v8$ and $v2$ in the S_1^1 output port. A star means that any frame from the corresponding input link can arrive at the same instant as $v0$ frame in the next output port. Additionally, Figure 5 gives the exact delay (ED) for each candidate scenario as well as an upper bound (UB) on the delay which will be detailed in next section. From Figure 5, the exact worst-case end-to-end delay for $v0$ is 154.64 μs (scenario $v8, v2, v6, v3$).

Note that the increase in the number of candidate scenarios is multiplicative with the number of input links and/or number of VLs per input link. For instance, a network configuration with only 62 VLs and having 2 VLs per input link will lead to 2^{30} scenarios. Thus, it becomes impossible to analyse all the scenarios in limited time and with finite memory space. Therefore, the MC approach in [7] cannot cope with industrial configurations including 1000 VLs. For that reason, a different approach has been proposed that computes upper bounds on the end-to-end delays rather than the exact delays.

C. Computing upper bounds on VL delays

The NC approach upper-bounds the delay of a VL at an output port h by computing the maximum horizontal distance between the overall arrival curve $\alpha_o^h(t)$ of VLs at node h and the service curve β^h(t) provided to these VLs by node h. Let us illustrate this delay bound computation for v_0 of the network example in Figure 1.

The service provided to all the VLs traversing an output port h with the link rate R is lower bounded by a service curve as $\beta^h = R[t - sl]^+$ where, $[a]^+$ means $\max\{0, a\}$ and sl is the switching latency. In the given network example, we assume sl = 0. So, the service curve at each output port is the same and is given as $\beta^{e_1} = \beta^{S_1^1} = \beta^{S_2^1} = 100[t]^+$

The overall traffic at the output port h is then given by an overall arrival curve $\alpha_o^h(t)$. The overall arrival curves integrating offset based scheduling from AFDX end-systems are proposed in [13]. We summarise the process.

The network configuration interconnects n_e end-systems $e_1, \ldots e_{n_e}$. Each of these end-systems e_k generates a (possibly empty) set V_k^h of n_k^h VLs sharing a given output port h. For example, the switch output port S_1^1 in Figure 1 is shared by five VLs scheduled by three end-systems: v_0 by e_1, v_1 and v_2 by e_4, v_8 and v_9 by e_5. So, the basic idea consists in building one arrival curve ($\alpha^h_{aggr,e_k}(t)$) for each end-systems e_k with a non-empty set V_k^h. This arrival curve is the maximum of the aggregated arrival curves of v_x in V_k^h. The aggregated arrival curve of v_x assumes v_x as the benchmark VL. It is the sum of the arrival curve of v_x at h and the arrival curves of the other VLs v_y in V_k^h shifted right by their relative offset to v_x. Then, the overall arrival curve ($\alpha_o^h(t)$) is the sum of per end-system arrival curves. Formulas are given in [13].

Let's illustrate the process on e_5. Two aggregated curves are built, one for each VL coming to S_1^1 from e_5. Figure 6 (top-left and top-right) shows aggregated curves $\alpha^{S_1^1}_{aggr,8}$ and $\alpha^{S_1^1}_{aggr,9}$ with v_8 and v_9 as benchmark VLs respectively. Then the arrival curve form the end-system is the maximum of these aggregated curves, as depicted in Figure 6 (bottom-left). Figure 6 (bottom-right) shows arrival curves for end-systems e_1, e_4 and e_5 at the S_1^1 output port, as well as the overall arrival curve (sum of the three previous curves) at the same port. This overall arrival curve is then used to compute the delay upper bound.

At S_2^1, the delay upper bound can be computed in a similar manner. The only difference is that the aggregated flows (from source end-systems e_1, e_4 and e_5) arriving through S_1^1 are now serialised. Indeed, they are buffered in the same FIFO queue in S_1^1. Thus they are emitted one by one on the link towards S_2^1-. In the following output ports, these frames remain serialised and cannot arrive at the same time. This serialisation is integrated in the NC [10] by considering that the arrival curve from one input port has a burst tolerance of no more than the largest frame size on the link and a rate not higher than the transmission rate of the link. Thus, the arrival curve of flows serialised at S_1^1 arriving at S_2^1 is the minimum between the arrival curve with the burst equal to largest frame size and with the rate equal to the link rate and the sum of the per end-system arrival curves.

The overall process follows a dataflow order. Then an upper bound on the end-to-end delay of a given VL is obtained by summing the upper bounds in the output ports on the VL path. For v_0 in Figure 1, the maximum delay upper bounds at e_1, S_1^1 and S_2^1 respectively are 8.56, 60.56 and 85.93 μsec. Thus, the end-to-end delay upper bound is 155.05 μsec. In Figure 5, this upper bound (UB) is shown at the root of the inverted tree, since it covers all the worst-case candidate scenarios.

The same process can be applied on a subset of worst-case candidate scenarios. As an example, let's consider node $v_8, v_1, *, *$ in Figure 5. It represents the subset of worst-case scenarios where a v_0 frame arrives in the output port S_1^1 at the same time as v_8 and v_1 frames. Therefore, the overall arrival curve at S_1^1 only considers this situation: the arrival curves for end-systems e_4 and e_5 are respectively the curves where v_1 and v_8 are the benchmark VLs. Conversely, node $v_8, v_1, *, *$ represents all worst-case candidate situations at the output port S_2^1: v_4, v_5, v_6 and v_7 can be the benchmark VLs coming from e_3. Thus, the arrival curve of e_3 takes into account four aggregated arrival curves. Overall, the upper bound on the end-to-end delay for v_0 is 144.03 μs for this subset of scenarios.

D. Problem Statement

MC computes exact delay but is limited to small network configurations, whereas NC can be easily scaled to industrial-size networks but computes pessimistic delay upper bounds.

Obviously the computation of the exact worst-case delay should be preferred. Indeed, an over estimation (i.e. upper

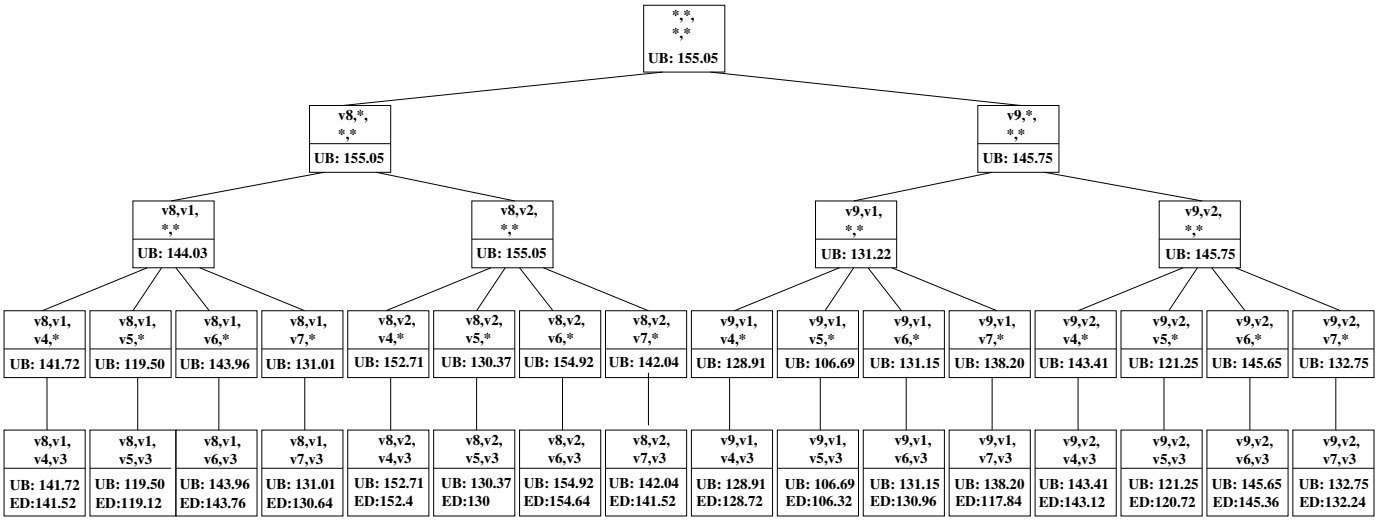


Fig. 5. Worst-case analysis of v_0

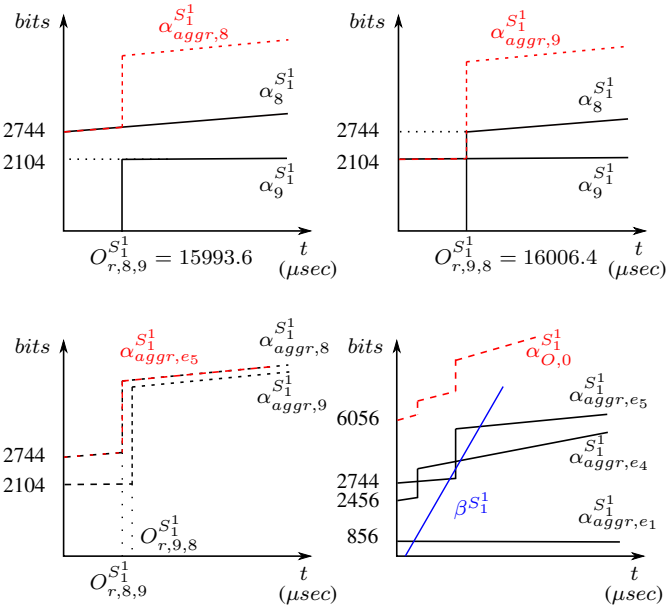


Fig. 6. NC curves at S_1^1

bound) of this worst-case delay can lead to over-dimensioning of the network in order to ensure that deadlines are met.

It is important to note that the computation time of both types of solutions is very different. MC approach suffers from the fact that adding more interfering VLs and/or switches in the path of the VL under study results in multiplication of the number of scenarios to be analysed. In this approach, the total number of possible scenarios at an output port is the multiplication of the number of frame sequences from each input link. For instance, coming back to worst-case analysis of v_0 in Figure 1, adding only 4 more VLs through an input at S_2^1 will increase the number of scenarios to 16×4 = 64 scenarios. Whereas in an industrial AFDX configuration there are as many as 1000 VLs which will lead to a huge number of scenarios to be analysed, which cannot be done in an acceptable time duration. Moreover, it would require a lot of memory to store and process the corresponding frame sequences for the given number of scenarios.

[7] shows that this approach computes exact worst-case delays in around 1 hour when up to 50 VLs are involved and within a reasonable limit of computation time (exact time is not specified) when at most 60 VLs are involved.

On the other hand the computation of an upper bound of the worst-case delay as presented in Section II-C requires a single data flow computation which is very fast (few milliseconds when 1000 VLs are involved). This computation can integrate every possible scenario – corresponding to the root of the inverted tree in Figure 5 – or a subset of these scenarios – corresponding to another node of the tree.

In this paper, the idea is then to build a hybrid approach that leverages the over approximation of the worst-case delay for a subset of the worst-case candidate scenarios to avoid the exact delay computation on every scenario in the subset. The principle can be illustrated on the analysis of v_0 for the configuration in Figure 1. As previously explained and depicted in Figure 5, 16 scenarios are candidate to the worst-case. Let's consider that the exact delays for the 8 scenarios in the left part of the tree (corresponding to the situation when v_8 delays v_0 in the output port S_1^1) have been computed. The worst-case delay among these scenarios is 154.64 μsec. Concerning the right part of the tree (v_9 delays v_0 in the output port S_1^1), the delay upper bound computed by the NC approach (Section II-C) is 145.75 μsec. It means that no candidate scenario in the right part of the tree has an exact delay that exceeds 145.75 μsec. Since one scenario in the left part of the tree has an exact delay of 154.64 μsec, computing the exact delay for the scenarios in the right part of the tree is useless.

In this specific case, the 16 exact delay computations can be safely replaced by 8 exact delay computations and one

upper bound computation. It represents a significant reduction of the number of computations and time needed to get the exact worst-case delay. In case of industrial-size network, if this reduction still leads to an unacceptable computation time, the process can be stopped at any time to find an upper bound which is tighter than the one provided by the NC approach.

In the next section we show how this principle can be generalised, leading to HA.

III. Hybrid Approach

The state-space of a given VL is the whole set of worst-case scenarios for this VL. As shown earlier, the delay upper bound on a subset of this state-space indicates whether the worst-case scenario can be in this subset or not. Therefore, HA identifies all the subsets that cannot include the worst-case scenario and eliminates them. Then the exact delay has to be computed only for scenarios in subsets which cannot be eliminated. This process involves the following steps:

- calculate the exact delay for one carefully selected scenario, using the computation presented in Section II-B,
- eliminate subsets with a delay upper bound (computed by the approach presented in Section II-C) smaller than the exact delay computed in the previous step,
- calculate the exact delay for one scenario in a non-eliminated subset and, if this delay is larger than the previous one, eliminate additional subsets,
- repeat the same process until, either a time limit has elapsed, or every scenario is in an eliminated subset or its delay has been computed.

A. HA Algorithm for exact worst-case delays or tighter upper bounds

An implementation of HA is given in Algorithm 1. It assumes no time limit and computes the exact worst-case end-to-end delay for each VL v_i in the network (line 1-9). Let us focus on the computation for v_0 in the AFDX network example in Figure 1. The worst-case delay for v_0 is initialised to 0 (line 2 in Algorithm 1). As shown in Figure 5, v_0 state-space contains 16 candidate scenarios (the tree leaves) derived from the tree root and 22 subset scenarios (the tree branch points). The first step is to compute the exact delay for one of the candidate scenarios. The choice of this first scenario highly impacts the performance of HA. For instance, the exact delay of scenario v_9,v_1,v_5,v_3 is 106.32 μsec. If it is selected no branch points (BPs) can be eliminated since they all correspond to a subset of scenarios having a delay upper bound greater than 106.32 μsec. We consider the following greedy heuristic: at each level in the tree we select the branch with the largest delay upper bound. Indeed, it is more likely that the leaf at the end of the branch with the highest delay upper bound will have a larger value of exact delay. The steps to determine this leaf and also to find the exact worst-case delay using HA are illustrated in Figure 7.

Starting from the root of the tree (line 3 in Algorithm 1), the delay upper bounds on the initial branch points are computed (Figure 7 (a)), 155.05 μsec for $v8$,*,*,* and 145.75 μsec

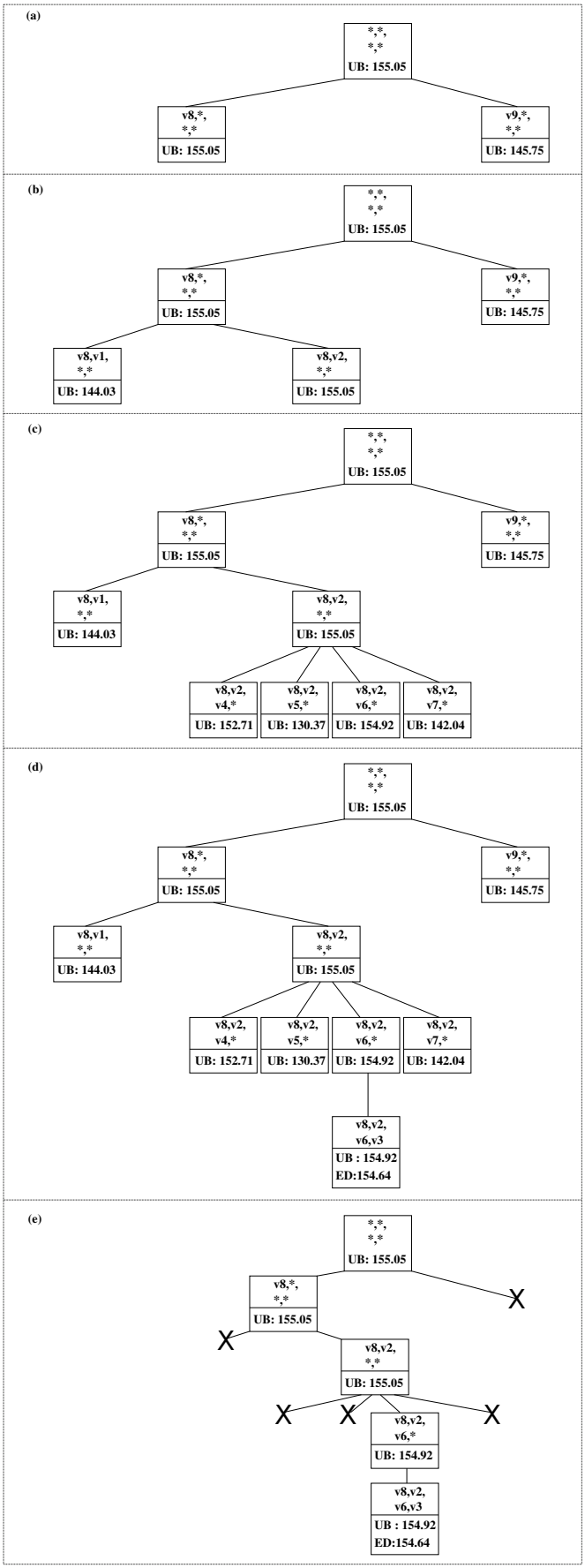


Fig. 7. Exact worst-case E2E delay computation of v_0 using HA

for $v9$,*,*,*.Therefore the branch point with highest upper bound is selected, i.e. $v8$,*,*,*. The same process is repeated (Figure 7 (b) and (c)) with selection of scenarios $v8$,$v2$,*,* (155.05 μsec) and $v8$,$v2$,$v6$,* (154.90 μsec). The selected branch terminates at leaf $v8$,$v2$,$v6$,$v3$. This selection process is performed by function *SelectLeaf()* (line 5) in Algorithm 1, where upper bounds (UBs) for 8 branch points are calculated to obtain the given leaf. The exact delay of 154.64 μsec is computed at this leaf (line 6), also shown in Figure 7 (d).

Next, this exact delay is compared to the delay upper bounds of all branch points previously computed. Indeed, these branch points cover all the leaves except the one where the delay has been computed. In this example, all these computed branch points (other than that of the selected leaf) have a delay upper bound less than 154.64 μsec. It means that the exact delays of their corresponding leaves will also be less than 154.64 μsec. So they are safely eliminated to obtain a reduced state-space (Figure 7 (e)). This process is performed by function *CompareDelays()* (line 9) in Algorithm 1. Since there are no more branch points left to be analysed, the process can be stopped (the loop in line 4 will stop) for v_0 and the exact worst-case delay is that of leaf $v8$,$v2$,$v6$,$v3$, which belongs to a scenario where v_0 is delayed by v_8 and v_2 at S_1^1 and by v_6 and v_3 at S_2^1. The same process is repeated in Algorithm 1 for the other VLs in the network configuration.

Algorithm 1: Hybrid Approach algorithm

Output: EWCD : per flow exact worst-case delay
Data: VLs : list of flows v_1, v_2 ...
Data: ED : exact delay
Data: UBs : list of upper bounds
Data: BPs : list of branch points
Data: root : tree root
Data: leaf : tree leaf
1 **foreach** $v_i \in$ *VLs* **do**
2 EWCD = 0;
3 BPs = root(v_i) ; `// get tree root node`
4 **while** *notAnalysed(BPs)* **do**
5 UBs, leaf = *SelectLeaf*(v_i, BPs);
6 ED = *ComputeExactDelay*(v_i, leaf);
7 **if** *(EWCD < ED)* **then**
8 EWCD = ED;
9 BPs = *CompareDelays*(v_i, UBs, ED);

SelectLeaf() returns the *leaf* and *UBs* corresponding to the branch having the highest upper bounds among the given branch points. It ignores the leaves analysed in previous iterations.
ComputeExactDelay() computes *ED* for the given leaf.
CompareDelays() compares *ED* and *UBs*, eliminates the branch points where UB $<$ *ED* and returns non-eliminated branch points.

One single exact delay computation has to be executed for v_0, since it allows eliminating every other worst-case candidate scenario. The situation might be less favourable. This can be illustrated with the analysis of v_4, as shown in Figure 8. v_4 shares switch output ports with VLs from four end-systems: v_8, v_9 from e_5, v_1, v_2 from e_4, v_3 from e_2 and v_0 from e_1. The most promising branch corresponds to the situation where v_4 arrives in S_2^1 at the same time as v_8, v_2, v_3 and v_0. The exact delay for this worst-case scenario candidate is 126.72 μs. It allows the elimination of the subtree at the branch point $v_9, *, *, *$ (UB = 124.14 μs) but, since UB for subtree $v_8, v_1, *, *$ is 126.92 μs, it has to be analysed.

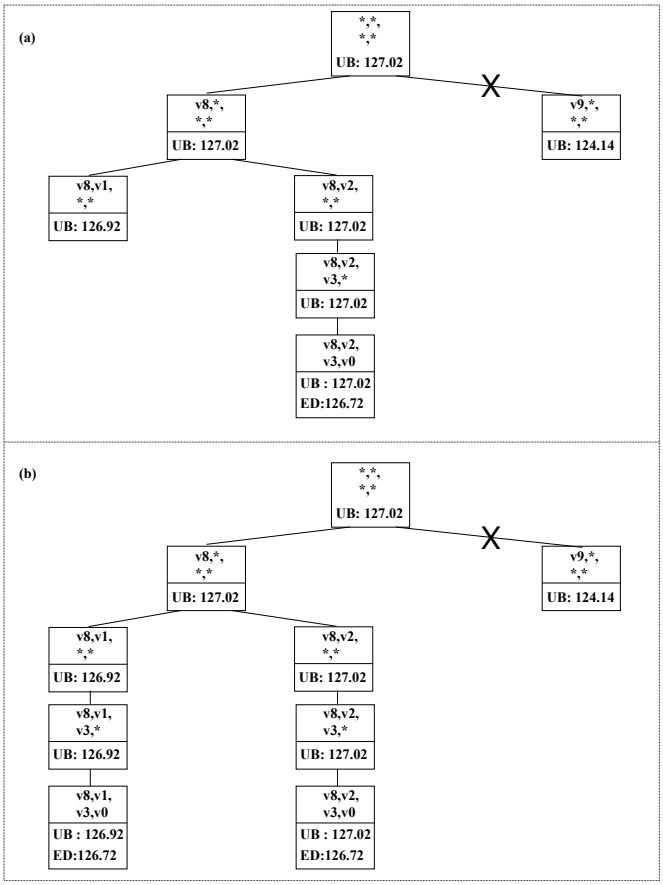


Fig. 8. Exact worst-case E2E delay computation of v_4 using HA

Considering an industrial configuration with around 1000 VLs, the reduction in the number of scenarios to be analysed in HA may be insufficient to complete the process in acceptable time. In such a case HA algorithm can be interrupted at any moment to obtain improved delay upper bounds. It can be illustrated on flow v_4 previously discussed. Let us assume that the process was interrupted right after calculating the exact delay of $126.72\mu s$ at the first selected leaf (v_8, v_2, v_3, v_0 in Figure 9). In this case, the delay of 126.72 μs does not represent the worst-case delay for v_4, however, it is the maximum delay that can be observed in the sub-branch $v_8, v_2, *, *$. This also means that the delay UB of 127.02 μs computed by NC is pessimistic, and it can be safely replaced by 126.72 μs. Similarly, at the branch point $v_8, *, *, *$ the UB is the maximum of its sub branches' UBs, i.e. $max\{126.92, 126.72\} = 126.92\mu s$. Finally, an improved delay UB of v_4 can be obtained at the tree root, which is the maximum UB among its sub branches $v_8, *, *, *$ and $v_9, *, *, *$,

i.e. $max\{126.92, 124.14\} = 126.92\mu s$.

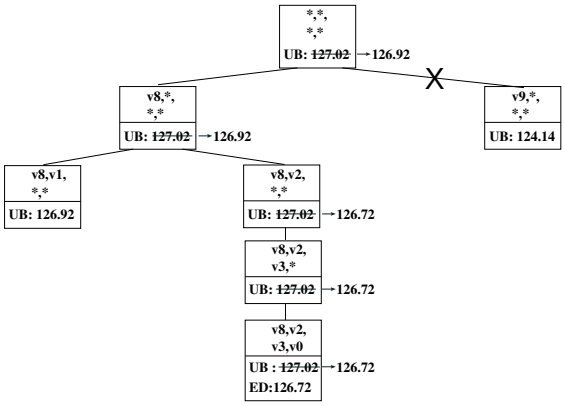


Fig. 9. Improved UB computation of v_4 using HA

B. Number of computations in HA

As illustrated by the above examples HA requires two sets of computations, one set of exact worst-case delay computations based on the approach presented in Section II-B and one set of upper bound computations using the approach presented in Section II-C. As previously mentioned the goal of HA is to limit the combinatorial explosion of the MC approach presented in Section II-B while still getting the exact worst-case delay. This limitation is reached thanks to a drastic reduction of the number of exact delay computations, at the cost of upper bound computations. In the following paragraph, we formalise the number of exact delay and upper bound computations for a VL analysis.

Let us start with the number of computations for the approaches presented in Sections II-B and II-C. The number of worst-case candidate scenarios depends on the number of VLs competing with the VL under study and the number of end-systems. As presented in Section II-C, each output port h is crossed by sets of VLs V_k^h, where end-system e_k is the source of the VLs in set V_k^h. The VL under study v_x follows a path p_x including pl_x switch output ports $h_1, \ldots, h_{pl_x}$. Therefore, every VL in a set V_k^h, with h belonging to path p_x, competes with v_x. Let's call V_{v_x} the set of VLs competing with v_x. We have:

$$V_{v_x} = \bigcup_{1 \leq k \leq n_e, h \in p_x} V_k^h \tag{1}$$

with

$$V_k^h = \{v_1^{k,h}, \ldots, v_{n_k^h}^{k,h}\} \tag{2}$$

In all existing avionics network configurations, two VLs that separate in a switch (are transmitted on different output links) will never meet later. Thus, $V_k^{h_i}$ and $V_k^{h_j}$ with $h_i \neq h_j$ are disjoint sets.

A tree such as the one in Figure 5 can be built for v_x. V_k^h sets are considered in an arbitrary order. In order to simplify the presentation, we rename these sets $V_1, V_2, \ldots, V_{nbs}$ with $V_i = \{v_1^i, \ldots, v_{n_i}^i\}$. The root of the tree represents any candidate worst-case scenario. It has n_1 children, one per V_1 element, labelled $v_{i_1}^1, *, \ldots, *$ with $1 \leq i_1 \leq n_1$. Each of these nodes has n_2 children labelled $v_{i_1}^1, v_{i_2}^2, *, \ldots, *$ with $1 \leq i_1 \leq n_1$ and $1 \leq i_2 \leq n_2$. The process repeats until the leaves of the tree which are labelled $v_{i_1}^1, \ldots, v_{i_{nbs}}^{nbs}$.

Therefore, the tree is composed of $n_1 \times n_2 \times \ldots \times n_{nbs}$ leaves and $n_1 + (n_1 \times n_2) + \ldots + (n_1 \times n_2 \times \ldots \times n_{nbs-1})$ branch point nodes, excluding the root.

The exact worst-case delay analysis presented in Section II-B computes the delay for each worst-case candidate scenario, i.e. each leaf of the tree. Therefore, $n_1 \times n_2 \times \ldots \times n_{nbs}$ computations are required.

The upper bound analysis presented in Section II-C requires one single computation, corresponding to the root of the tree that encompasses all worst-case candidate scenarios.

HA will compute the exact delay for between n_{nbs} and all worst-case candidate scenarios and the upper bound for a subset of branch point nodes. More precisely, the count can be divided in two parts.

- In the first part, the delay for the most promising worst-case candidate scenario is computed. To that purpose the upper bound is computed for the n_1 children of the root node. Then the upper bound is computed for the n_2 children of the node with the highest upper bound, and so on until the leaves. It leads to $n_1 + \ldots + n_{nbs-1}$ upper bound computations and n_{nbs} exact delay computations.
- In the second part, additional upper bound and exact delay computations are done for the part of the tree with an upper bound larger than the current worst-case delay.

Coming back to the network example in Figure 1, in HA the exact worst-case delay for v_0 is obtained after a total 9 computations: 1 leaf for the exact delay (out of 16 leaves) and 8 branch points for upper bounds. The results of HA for all the VLs in Figure 1 are summarised in Table II. It can be observed, for all the VLs, that the total number of exact delay computations (EDS) required in HA is significantly low with respect to the total number of worst-case candidate scenarios (MC state-space), at the cost of few upper bound computations (UBS).

As mentioned earlier, the overhead of UB computation, in terms of execution time, is negligible as compared to the gain obtained by reducing the exact delay computation scenarios. This gain in execution time is not apparent on the small network example of Figure 1 but, as shown in the next section, it is very large for an industrial-size network configuration.

Also note that the difference between EWCD and UB in Table II is very small. But, this difference (i.e. pessimism of NC) on the industrial configuration is about 8% on average and higher than 40% for more than 10% of the flow paths.

IV. HA evaluation on a Real Avionics Configuration

The HA approach has been implemented using C++ programming language. The results presented in this paper are obtained on a Core i5-4210U processor (2.4 GHz) utilising a single processor thread with 32 Gb memory. HA is compared to the state-of-the-art MC (presented in Section II-B) and NC

TABLE II
HA ON VLS IN FIGURE 1

VL	max UB	EWCD	MC state-space	EDS	UBS
v_0	155.05	154.64	16	1	8
v_1	149.15	148.88	8	1	6
v_2	171.05	170.64	8	1	6
v_3	98.33	97.92	16	3	12
v_4	127.03	126.72	4	2	4
v_5	82.5	81.92	4	2	4
v_6	131.5	131.20	4	2	4
v_7	105.48	104.96	4	2	4
v_8	173.9	173.52	8	1	6
v_9	158.18	157.84	8	1	6

UB = delay upper bound. EWCD = exact worst-case delay
UBS = upper bound scenarios. EDS = exact delay scenarios

(presented in Section II-C) approach. The comparison considers a realistic network configuration based on the network architecture of the A380 aircraft.

This network configuration includes 96 end-systems interconnected by 8 switches forwarding 984 VLs on 6276 paths (multicast VLs). Each VL is constrained by a frame length between 84 and 1555 bytes and a minimum inter-frame duration (BAG) between 2 and 128 ms. Table III (left and centre) shows the dispatching of VLs among BAGs and maximum frame lengths. Table III (right) shows the distribution of paths lengths (number of crossed switches).

TABLE III
VL AND PATH CHARACTERISTICS IN INDUSTRIAL CONFIG

BAG	VLs
2	20
4	40
8	78
16	142
32	229
64	220
128	255

l^{max}range	VLs
84–200	278
201-400	396
401–600	157
601–900	69
901-1200	28
1201–1500	51
>1500	5

Length	Paths
1	1780
2	2807
3	1436
4	253

A. Exact worst-case delay computation using HA vs MC

In [3], an evaluation of MC approach on a similar industrial configuration shows that it can successfully compute the exact worst-case delay, within 1 hour of computation time, on a VL path when there are at most 50 concurrent VLs, but no precise information about the state-space is given.

So, we compared our HA approach to the MC approach on different paths under the same conditions, i.e. at most 1 hour of computation time per path. The obtained results are summarised in Table IV and Figure 10. Out of 6276 paths, MC concluded for only 1209 paths, whereas, HA competed for 465 more paths (i.e. total 1674 paths).

A detailed analysis of these results is presented in the following paragraphs. First we compare the number of scenarios which have to be computed by both HA and MC for each VL path. Second, we compare the execution time for both approaches.

1) Comparing the number of scenarios: We recall that MC approach only computes ED for candidate worst-case scenarios, while HA additionally computes UBs for sets of scenarios. Therefore, the state-space is the number of ED computations for both approaches, increased by the number of UB computations for HA.

Paths are divided based on the size and computation time of the MC state-space in Table IV. The first 819 paths have an MC state-space smaller than 10^4 and include 6 to 89 concurrent VLs. On average there are 2.1×10^3 (ED) scenarios in MC. Whereas, the state-space of HA has about 7×10^2 (ED+UB) scenarios. This means the HA state-space is about 67% smaller. Similarly, on the next 390 paths, the HA state-space is about 49% smaller. On these $819 + 390 = 1209$ paths the computation time per path is less than 1hr for both MC and HA approaches. On the remaining, 465 paths the computation time of MC approach is much larger, either due to a very large state-space or due to a large number of concurrent VLs. On these paths, when the MC state-space is between 10^4 and 10^6, the HA state-space is 91% smaller. For the largest MC state-space the HA state-space is 92% smaller. On average the HA state-space is about 87% smaller for all the 1674 paths.

The relative size difference of the state-space on individual paths can be observed in Figure 10 (top). The state-space of MC is the reference and is normalised to 100%. The relative state-space of HA is given by:

$$\frac{\textit{HA state-space}}{\textit{MC state-space}} \times 100$$

Recall that the advantage of HA is the reduction of the number of ED computation. So, in Figure 10 (top) the state-space of HA is separated between ED computations (green, orange or red bars) and UB computations (light grey bars). For illustration purpose, VL paths are sorted in decreasing order of the absolute difference in MC and HA state-space, i.e. MC state-space - HA state-space. The paths where the number of ED computations in HA is smaller than that of MC are shown by green bars and the remaining paths (no reduction in ED computations) are shown by orange or red bars.

1128 out of the 1674 analysed paths benefits from HA, i.e. the number of ED computations is significantly reduced: the average reduction is 88% (green bars in Figure 10 (top)). For the remaining 546 paths, the number of ED computations is the same for both approaches. Since the HA state-space includes an overhead of UB computations, in some paths the HA state-space is larger than that of MC. There are 257 paths where HA has no reduction in ED computations and some UB computation overhead that accounts for on average 3.45% larger state-space (red bars in Figure 10 (top)). On the other 289 paths, there is no gain/loss in HA (orange bars in Figure 10 (top)).

On the paths where HA has reduced ED computations but the state-space is still larger due to the UB computation overhead, the gain/loss in HA can only be judged based on the computation time. Indeed, the computation time of UBs is

TABLE IV
HA COMPARED TO MC

MC State-Space	Paths analysed	Concurrent VLs	Average State-Space			Execution Time			
			MC	HA	% Smaller	Per Path		Average	
						MC	HA	MC	HA
$< 10^4$	819	6 – 71	$2.1 * 10^3$	$7 * 10^2$	**67%**	0sec – 54min	0sec – 49sec	5min	**3sec**
10^4 – 10^6	390	36 – 89	$1.4 * 10^5$	$7.2 * 10^4$	**49%**	13sec – 59min	0sec – 45min	17min	**4min**
10^4 – 10^6	382	42 – 100	$1.8 * 10^5$	$1.6 * 10^4$	**91%**	+1hr	0sec – 59min	NA	**1min**
10^6 – 10^7	83	53 – 106	$4 * 10^6$	$3.1 * 10^5$	**92%**	+1hr	90sec – 54min	NA	**23min**
Overall	1674	6 – 106	$2.77 * 10^5$	$3.68 * 10^4$	**87%**	-	0sec – 59min	NA	**3min**

much faster than the one of EDs. So, HA can still outperform MC even when HA state-space is larger.

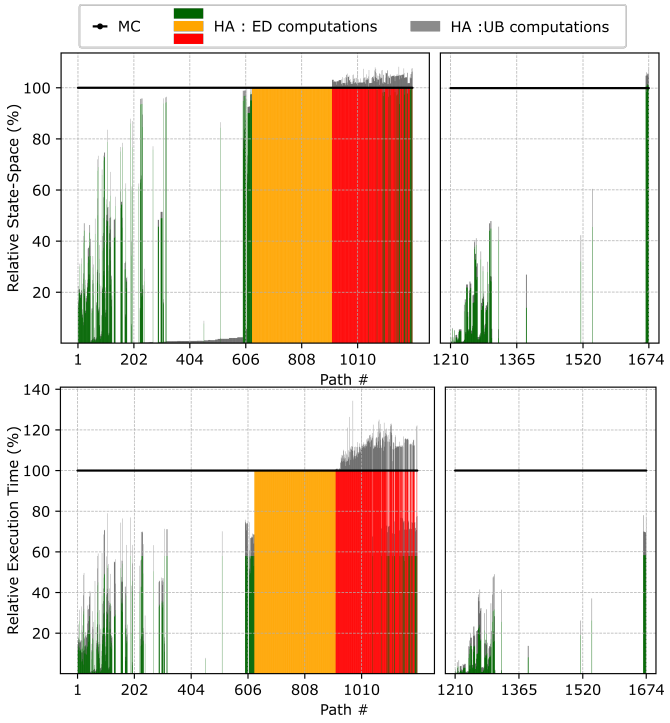


Fig. 10. Relative gain/loss of HA for an industrial configuration
Note : Green, Orange and Red bars correspond, respectively, to the paths where HA has positive gain, no gain/loss and some small loss in state-space/execution.

2) Comparing the Execution Time: The execution time comparison between HA and the MC approach is also shown in Table IV and Figures 10 (bottom).

For the comparison scenarios in Table IV, the HA approach outperforms MC.

For the 1209 paths on which MC was able to conclude within 1 hour, the average computation time per path is between 5 and 17 minutes for MC and between 3 sec and 4 minutes for HA. This improvement is mainly due to the reduction in number of ED computations in HA but also due to the faster computation of UBs in the HA state-space. On the remaining 465 paths, HA was able to conclude within 1 hour, but MC could not.

It can be observed from Figure 10 (bottom) that the relative computation time of the HA approach on individual paths is significantly less as compared to MC approach. For comparison purpose, the paths in Figure 10 (bottom) and Figure 10 (top) are sorted in the same order and follows the same colour code for ED and UB computations. However, in Figure 10(bottom), all the paths where the computation time of the HA state-space is higher than that of MC are marked red.

Not surprisingly, all the 1128 paths where the ED computations are reduced have significantly smaller computation time in HA: on average 92% faster. On 289 paths, HA and MC have the same computation time. On the remaining 257 paths the HA execution time is larger (about 12%), due to no reduction in number of ED computations and/or due to too large UB computation overhead.

3) Extended evaluation of exact worst-case delay computation using HA: The comparison scenarios discussed in the previous paragraphs yielded results only for the paths with the MC state-space up to 10^7 and with up to 106 concurrent VLs. On these paths, the computation time was limited to 1 hour. An extended analysis is carried out to evaluate the performance of the HA approach on other paths where the computation time is larger, either due to a very large state-space or a large number of concurrent VLs. The results are summarised in Table V. Total 233 paths have been analysed. The MC state-spaces consisting of up to 10^{10} ED scenarios are observed on these paths, on the contrary HA has a much smaller state-space (up to 10^7 scenarios). The gain in the HA approach is calculated only in terms of the reduction in state-space. Indeed, for these paths MC approach is not expected to conclude, even with an execution time of a month.

TABLE V
EXTENDED HA EVALUATION

MC State-Space	Paths	Conc. VLs	Average State-Space		
			MC	HA	% Smaller
10^4–10^6	69	63–100	$6.3 * 10^5$	$6.2 * 10^5$	**1.5%**
10^6–10^8	157	55–114	$7.9 * 10^6$	$2.9 * 10^6$	**62%**
10^8–10^{10}	7	71–118	$1.6 * 10^8$	$1.1 * 10^7$	**92%**
Overall	233	55 – 118	$1.04 * 10^7$	$2.5 * 10^6$	**76%**

B. Delay upper bound computation using HA vs NC

As discussed in Section IV-A, HA has an advantage of being an anytime algorithm, which means it can be interrupted at any moment to calculate the delay UB. In addition to the 1674 paths discussed above, where the exact worst-case delay was obtained, we computed delay UB on the remaining 4602 paths using HA by limiting the computation time to 1 minute. The results are compared to the UB computed using NC.

For comparison purpose, we also calculated a delay lower bound (LB) on all the paths using an optimistic method described in [14]. Indeed, a delay lower bound is always less than or equal to the exact-worst case delay, and allows quantifying the upper limit on the pessimism in the delay UBs. Figure 11 (top) shows the comparison between UBs obtained using HA (orange dots) and NC (blue dots) and the LBs (green line), where the paths are sorted in decreasing order of LBs for illustration purpose. Not surprisingly, the HA UBs are always smaller than or equal to those of NC on all the paths, thus, they are less pessimistic. The pessimism of these UBs is also shown in Figure 11 (middle and bottom), and is calculated for each path as:

$$\%\ pessimism = \frac{UB - LB}{LB} \times 100$$

The average and maximum pessimism of NC UBs are 8.02% and 59.75%, respectively. The HA was able to compute about 43% less pessimistic bound, the remaining pessimism of these UBs is on average 4.53% and the maximum pessimism is of 17.86%.

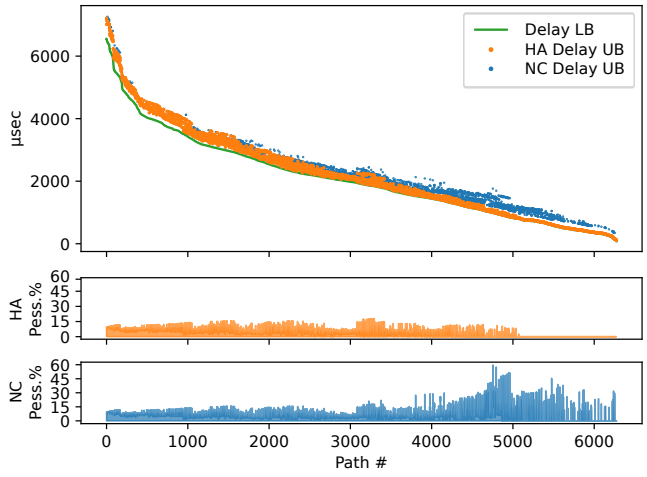


Fig. 11. End-to-end delay upper bound improvement in HA

C. Discussion

Results on the realistic case study show that the HA approach outperforms MC in terms of execution time and clearly mitigates the memory problem of MC by reducing the state-space. Notably, HA allows computation of exact delays on an industrial-size network where the state-space is too big to be analysed by MC approach.

However, the gain of HA highly depend on the characteristics of the analysed VL path. Since HA eliminates subsets when their upper bound is smaller than the currently computed largest exact delay, the number of eliminated subsets increases when upper bounds are very different among internal nodes. Conversely, subsets are hardly eliminated when upper bounds are similar among internal nodes.

On the example in Figure 7, one single worst-case candidate scenario is analysed, since computed upper bounds for subsets of candidate scenario significantly vary. One reason for this variation is the difference of frame size between VLs. To illustrate the impact of frame sizes on the variation of upper bounds, let us consider the example in Figure 1 with identical frame size among VLs, i.e. 339 bytes. In this situation the worst-case delay for v_0 is 189.84 μs and the upper bounds for each subset of candidate scenarios is given in Figure 12. Since every upper bound is larger than the overall worst-case, no subset can be eliminated. Therefore, every worst-case candidate scenario has to be analysed.

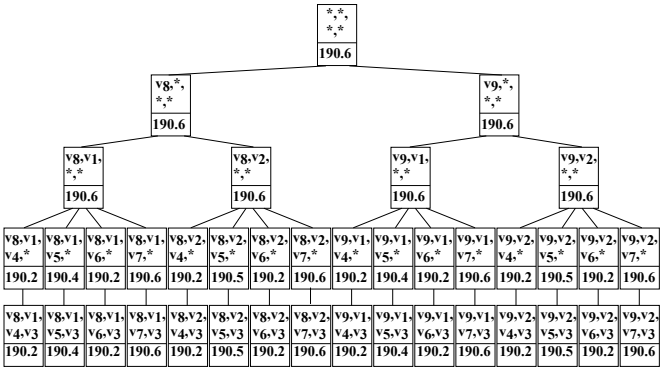


Fig. 12. Upper bounds with identical frame sizes

On the given industrial configuration, very few such cases are observed. They correspond to paths with the red bars in Figure 10.

V. Conclusion

This paper presents and evaluates HA, a hybrid approach for the computation of exact worst-case delays and delay upper bounds on industrial real time Ethernet networks. HA leverages the upper bound computation based on NC in order to mitigate the combinatorial explosion of the exact worst-case computation based on MC. It also leverages exact delay computation in order to tighten bounds on worst-case delays. Results show that HA scales up to an industrial case study. The evaluation considers an avionics network. Nevertheless HA can cope with any industrial context, provided that end node scheduling introduces offsets between flows. In the paper we assume the classical TDMA solution for avionics configuration. HA can be used with any other solution that leads to such offsets.

HA assumes a single traffic class. Therefore, frames are transmitted in a FIFO order in every output port. Such an assumption is not valid when flows with different criticality

levels share the same network. This situation will occur if additional less critical flows are transmitted on the AFDX network. Indeed, the impact of these additional flows will have to be bounded. It should be obtained thanks to service disciplines such as Priority Queueing or Round Robin, e.g. [18]–[22], or the promising IEEE TSN solution [23]. HA could be extended to such solutions, provided that an upper bound computation as well as an exact worst case one exist. These computations have to take into account offsets between flows emitted by a given end system. Such upper-bound computations exist, e.g. for Round Robin [24]. Up to now, exact worst-case computation is limited to FIFO.

Switch or link failure can lead to frame loss. It can be mitigated by frame replication. In avionics networks, source end systems transmit each frame on two separate paths and destination end systems keep the one received first. TSN proposes FRER which allows multiple frame replications and eliminations. The exact worst-case computation in HA requires that two frames that separate in a switch will never meet later (in another switch). It holds for current avionics replication solution. It will not be the case with FRER.

REFERENCES

[1] H. Charara, J.-L. Scharbarg, J. Ermont, and C. Fraboul, "Methods for bounding end-to-end delays on an AFDX network," in *18th Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE Computer Society, 2006.

[2] M. Adnan, J.-L. Scharbarg, and C. Fraboul, "Minimizing the search space for computing exact worst-case delays of AFDX periodic flows," in *6th IEEE International Symposium on Industrial and Embedded Systems (SIES)*, 06 2011.

[3] M. Adnan, J.-L. Scharbarg, J. Ermont, and C. Fraboul, "Model for worst case delay analysis of an AFDX network using timed automata," in *IEEE 15th Conference on Emerging Technologies Factory Automation (ETFA)*, 09 2010, pp. 1–4.

[4] ——, "An improved timed automata model for computing exact worst-case delays of AFDX periodic flows," in *IEEE 16th International Conference on Emerging Technologies Factory Automation (ETFA)*, 09 2011, pp. 1–4.

[5] ——, "An improved timed automata approach for computing exact worst-case delays of AFDX sporadic flows," in *IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA)*, 09 2012, pp. 1–8.

[6] R. Alur and D. Dill, "A theory of timed automata," in *Theoretical computer science*, vol. 126, no. 2. Elsevier, 1994, pp. 183–235.

[7] M. Adnan, *Exact worst-case communication delay analysis of AFDX network*. Ph.D. Thesis, 11 2013.

[8] J.-Y. L. Boudec, "Application of network calculus to guaranteed service networks," in *IEEE Transactions on Information Theory*, vol. 44, no. 3, 1998, pp. 1087–1096.

[9] S. Martin and P. Minet, "Holistic and trajectory approaches for distributed non-preemptive fp/dp* scheduling," in *4th International Conference on Networking (ICN)*, 2005.

[10] H. Bauer, J.-L. Scharbarg, and C. Fraboul, "Improving the worst-case delay analysis of an AFDX network using an optimized trajectory approach," in *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, 11 2010.

[11] N. Benammar, F. Ridouard, H. Bauer, and P. Richard, "Forward end-to-end delay for afdx networks," in *IEEE Transactions on Industrial Informatics*, vol. 14, no. 3, 03 2018, pp. 858–865.

[12] D. Thiele, P. Axer, and R. Ernst, "Improving formal timing analysis of switched Ethernet by exploiting FIFO scheduling," in *52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015.

[13] X. Li, *Worst-case delay analysis of real-time switched Ethernet networks with flow local synchronization*. Ph.D. Thesis, 09 2013.

[14] A. Soni, X. Li, J.-L. Scharbarg, and C. Fraboul, "Work in progress paper: pessimism analysis of network calculus approach on AFDX networks," in *IEEE 12th International Symposium on Industrial Embedded Systems (SIES)*, 07 2017.

[15] "Aircraft data network, parts 1,2,7 aeronotical radio inc." ARINC Specification 664, Tech. Rep., 2002 - 2005.

[16] O. Hotescu, K. Jaffrès-Runser, J.-L. Scharbarg, and C. Fraboul, "Multiplexing Avionics and additional flows on a QoS-aware AFDX network," in *24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019.

[17] X. Li, J.-L. Scharbarg, and C. Fraboul, "Improving end-to-end delay upper bounds on an afdx network by integrating offsets in worst-case analysis," in *IEEE 15th Conference on Emerging Technologies Factory Automation (ETFA)*, 09 2010, pp. 1–8.

[18] L. Lenzini, E. Mingozzi, and G. Stea, "Aliquem: a novel DRR implementation to achieve better latency and fairness at O(1) complexity," *Tenth IEEE International Workshop on Quality of Service*, 2002.

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

[20] M. Katevenis, S. Sidiropoulos, and C. A. Courcoubetis, "Weighted round-robin cell multiplexing in a general-purpose atm switch chip," in *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 8, 10 1991, pp. 1265–1279.

[21] H. M. Chaskar and U. Madhow, "Fair scheduling with tunable latency: a round-robin approach," in *IEEE/ACM Transactions on networking*, vol. 11, no. 4, 2003, pp. 592–601.

[22] H. Shimonishi, M. Yoshida, R. Fan, and H. Suzuki, "An improvement of weighted round robin cell scheduling in atm networks," *IEEE Global Telecommunications Conference (GLOBECOM). Conference Record*, vol. 2, pp. 1119–1123, 1997.

[23] M. Ashjaei, L. Lo Bello, M. Daneshtalab, G. Patti, S. Saponara, and S. Mubeen, "Time-Sensitive Networking in automotive embedded systems: State of the art and research opportunities," *Journal of Systems Architecture*, vol. 117, p. 102137, 2021.

[24] A. Soni and J.-L. Scharbarg, "Deficit round-robin: Network calculus based worst-case traversal time analysis revisited," in *LCN*, 09 2022, pp. 275–278.